

1. [Introduction to the course in MathML](#)
2. [Preparing for writing MathML](#)
3. [Displaying mathematical content](#)
4. [MathML infrastructure](#)
5. [Presentation token elements \(mi,mn\)](#)
6. [Presentation token element \(mo\)](#)
7. [Presentation token elements \(others\)](#)
8. [Presentation layout elements \(mrow,mfrac,msqrt,mroot\)](#)
9. [Presentation scripting elements](#)
10. [Presentation table elements](#)
11. [Writing MathML](#)
12. [Speeding up with MathML](#)
13. Appendices
 1. [Unicode character set \(A-H\)](#)
 2. [Unicode character set \(I-Q\)](#)
 3. [Unicode character set \(R-Z\)](#)

Introduction to the course in MathML

The encoding of mathematics in a document has two perspectives. One is to present mathematical content in visual form; other is to convey mathematical content in a form that can be interpreted by other programs (software) for use in a mathematical context.

The intent of the first is obvious and clearly understood. The second perspective is more complex and is targeted to create a language structure that can be used to exchange mathematics and its underlying concepts and ideas between different programs and systems. For example, MathML markups may result in a audio rendering system, which can read a mathematical expression like $\frac{dy}{dx}$: “...it is a differentiation of first order of ‘y’ with respect ‘x’“ or a program that could intelligently interact with the content coded in MathML and produce results or analysis.

In accordance with the intent, the MathML has two different encoding paradigms. The first caters to the requirement of presentation and the other that of content. Here in the course, we restrict ourselves with presentation mark ups as this course is set out for beginners and who seek to write MathML for display purposes.

Why MathML?

Let us have a look at a piece of encoding required to create a simple table (3 X 3) :

Example:

Typical code lines MathML

```
<m:math display="block"> <m:htable frame="solid"
columnlines="solid" rowlines="solid"> <m:mtr>
<m:mttd> <m:mi>x</m:mi> </m:mttd> <m:mttd>
<m:mi>y</m:mi> </m:mttd> <m:mttd> <m:mi>z</m:mi>
</m:mttd> </m:mtr> <m:mtr> <m:mttd> <m:mn>1</m:mn>
</m:mttd> <m:mttd> <m:mn>2</m:mn> </m:mttd> <m:mttd>
```

```
<m:mn>3</m:mn> </m:mtd> </m:mtr> </m:mtable>  
</m:math>
```

The display looks like :

x	y	z
1	2	3

Looking at the above MathML encoding, we observe that codes are simply bulky and repetitive in nature. This brings out the question : why to encode MathML, while we could have simply taken the snapshot of the mathematical content and put the same in light weight picture format like jpeg or gif ! The reasons, that this approach is not favored, are :

1: The picture files, even when light, would run in 100s of kilobytes, while the text part of the document would be hardly 15 – 20 Kilobytes. This is a significant design problem – particularly for web rendering of courses like this one. If this module is interspersed with pictures for mathematical expressions, then the module would take much longer (10 to 15 times) to download; and the readers would be really irritated if the document consisted of cross references to other such documents, containing mathematics in picture format.

2: It would be difficult and ugly to present mathematical content as part of the text, because picture renders with white space around itself. On the contrary, MathML allows to insert mathematical content, like x^5 , right inside the text paragraph in an elegant manner.

3: It would not be possible to copy and paste the content and modify the same. This constitutes a major handicap in treating mathematical content as picture. We can only select the picture file as a whole - not a part of it. (Try to copy the right hand expression of the equation presented below.)

$$(\alpha + \beta)^2 = \alpha^2 + \beta^2 + 2\alpha\beta$$

4: The problem with pictures is that they have an invisible box around them whose color remains that of the background at the time of its capture as picture. The color of the box is, thus, independent of background color of the rendering document. When the background color of the document (rendering system) is changed, mathematical content in picture format pops up with the color of box surrounding it.

The display of the same mathematical content as rendered with MathML encoding is presented here for comparison with the earlier display as picture :

Display with MathML :

$$\alpha^2 + \beta^2 + 2\alpha\beta$$

Nature of mathematical expression

Mathematical content or expression differs significantly in comparison to normal text. We here present the major differences in the display requirement of the two types of presentation.

Mathematical expression is vastly different in the visual presentation and associated meaning that it conveys. Rendering of normal text in the newspaper or printed books or the ones typed in the standard text editor on the desk top - thanks to the beauty of the structure of the language - is limited to, may be, 50 characters. A change in font and style does not change the meaning of the text (for example what is said in a paragraph), whereas, mathematical expression is sensitive to the visual forms of font and accompanying style. A particular character style may be attached to the underlying mathematical meaning. For this reason, there are about 900 Unicode characters used in MathML – many of which may not be required in a particular context, but needed in the totality of mathematical expanse.

Normal text flows in horizontal rows. Each character, composing the text, comprises of a base line, which is used as reference for horizontal alignment. In the case of mathematical expressions, the bases of the constituent characters are not at the same level. Consider the limits assigned to an expression involving integration.

$$\int_a^b e^x dx$$

This difference in the structural levels of constituents may be repetitive as well :

$$g(z) = e_{-\sum_{i=0}^{\infty} z_i^2}$$

The examples shown above highlight the two dimensional aspect of mathematical rendering. In the nutshell, normal text has a linear layout, whereas mathematical expression has two dimensional lay out. This aspect, in turn, introduces the requirement of alignment of characters along multiple horizontal and vertical lines.

Renderdering of expresion in the P/Q format is typical of the requirement of presentation in vertical direction. Presentation of $x = P/Q$, on the other hand, is actually a combination of single and two dimensional presentations, where linear ($x =$) and two dimensional elements (nominator and denominator parts above and below horizontal divider) are required to be aligned along a central horizontal line.

$$x = \frac{a}{b}$$

Further, repeated use of P/Q form in the vertical direction involves alignment along multiple horizontal lines :

$$x = \frac{\frac{(a+b)}{(c+d)} + \frac{(e+f)}{(g+h)}}{\frac{(i+j)}{(k+l)}}$$

The MathML provides a simple logical construct to encapsulate these complex presentation requirements of mathematical content. Fortunately, this logical construct is linear in that we may proceed to write codes starting from left of the expression to the end of it. This is one important characteristic of mark up paradigm in MathML. Even when handling two dimensional structures, we put the element designed for two dimensional layout mark up at the center of coding theme for that particular term and proceed to write code linearly for vertical components of display like numerator and denominator. For example, expression $\frac{dx}{dt}$, is coded with “mfrac” element which provides the frame work to write two dimensional expression.

Apparently, going by the bulk of the code writing; writing MathML appears a difficult proposition, but thanks to the mark up design team at W3C, we would actually not require more than two dozens such mark up elements for most of the situations - out of which only 10 elements would be in regular use. This is the simplicity of the paradigm provided in MathML that must always be kept in mind.

MathML and XML

MathML is a XML specification targeted to present mathematical content. In that sense, MathML may be considered as a subset of XML document. Mathematical content is usually placed with normal text, figures and other elements of a document. For this reason, the MathML encoding leaves normal text/ picture rendering to the parent XML environment (DTD). As such, there is a document specifications in the corresponding Document Type Definition (DTD) file for XML and a document specifications in the corresponding DTD file for MathML.

As XML encompasses MathML, it is logical that the syntax and grammar of coding or marking up in MathML is similar as that of XML document. The syntax and grammar for the use of elements of both XML and MathML in a document are largely governed by the definition contained within XML DTD, except for few additional rules defined by MathML DTD. On the other hand, MathML provides the specific details of the functioning of

MathML elements and few additional rules related to syntax and grammar for MathML markings.

For reasons as detailed above, the document that includes MathML should contain DTD declaration for both XML and MathML. The top most structure level of the document is defined by XML's "document" and "content" elements. The MathML codes are then inserted between the opening and closing tags of "content" element.

A typical structure of a "xml" document containing MathML elements would look like :

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?> <!DOCTYPE math PUBLIC
"-//W3C//DTD MathML 2.0//EN"
"http://www.w3.org/Math/DTD/mathml2/mathml2.dtd" >
<document
xmlns:m="http://www.w3.org/1998/Math/MathML">
<content> MathML code lines can be placed here.
..... </content> </document>
```

As pointed out, the MathML specification, in addition to providing details about its element, also defines additional syntax and grammar for MathML markings. First additional rule pertains to restriction on the assignment of the values to the attributes of the MathML element (say only a positive value). XML does not put this kind of restriction. The second additional rule pertains to the different treatments given to the child elements placed within a MathML element. Its order and uses are defined by MathML DTD. In MathML, we would require to place child elements in certain order to satisfy the requirement of necessary arguments in a particular order.

In the nutshell, we can conclude that syntax and grammar is largely defined by XML's DTD; whereas details of MathML elements and few additional syntax and grammars are defined by MathML DTD.

Writing in mathML

Writing codes in MathML appears complex, but it is not so in practice when taken up with certain basic planning. In particular, following facts that makes this task easy are :

- (i) Coding paradigm in MathML is linear which proceeds from left to right.
- (ii) There are about 25 mark up elements to master to write MathML for most of the situations - out of which only 10 may be regularly used.
- (iii) There are ways around via copy (ctrl+c) and paste (ctrl+v) to replicate blocks of codes.

A Foot note : why this course?

I was initiated to write a physics course for higher secondary school students on the connexions platform. It appeared to be a great place to write modules, courses and books. The concept of publishing and sharing knowledge without going into the trouble of finding a publisher was heaven-sent. I started with great energy and enthusiasm till I hit the road block of handling mathematics for my course.

There exists great many helps, tutorials, guides and “go around” suggestions splattered on the web site for using MathML, but it looked as if I needed a full dress down on this medium of mathematics for web publishing. By all counts, it appeared to be a real pain, constantly taking my attention away from the basic task of writing book/course on physics. I was just not satisfied with “work aorunds” - always fearing I might eventually run in to the dead end - right when I had finished creating bulk of the material for the course.

Eventually, what I wanted that I be sure of expressing MathML quickly and effectively. To this end, I have conjured up this tutorial that takes the help of familiar softwares available on the desktop to implement MathML.

In this course, we can begin with what is available on the desk top and what we had been using all along. Just we need to download a browser supporting MathML. In my case, the Internet explorer, on the desktop was out of the synch and so I downloaded Mozilla Firefox (easy to download

from : <http://www.mozilla.org>). The newer version of IE also supports MathML. There is no preference. It is just a question of what you have. Then, using the MS Word - the standard text editor available on most desktops - we can experiment through mathematical constructs and quickly acquire sufficient skills to go ahead with the electronic publication.

I would, however, be careful to spell out the limits on the coverage. Let us be clear that this course is not intended to be a comprehensive treatment on MathML. Instead, the coverage is restricted to areas, which form the basic part of the electronic publication of mathematical content. For this reason, not-so-useful details of attributes, parameters, options, tags, elements etc are simply spared. Further, the style of presentation is result oriented – not “teaching type”, and is solely based upon the perception of difficulties in presenting mathematics on web and its solution in using MathML.

It is my sincere endeavor to help authors, who feel the same about MathML and beginners who want to understand MathML paradigm in basic form. I am looking forward to [views and ways to improve](#) this course.

Preparing for writing MathML

This course heavily relies on practicing codes in MathML. For this reason, a frame work enabling practice in desktop environment is planned, which does not require any specialized tool or software. The requirement for practicing MathML is two fold : (i) a text editor that composes code lines as plain text without introducing any formatting characters and (ii) a browser that is capable to implement MathML.

It is recommended to use an advanced text editor like MS Word to handle the first task. We shall use it in a specialized mode to serve our purpose. Use of advanced text editor, as against editors designed to save file as plain text like notepad available on Windows platform, is recommended. This preference is mainly owing tot to the advanced editing facilities available and the ease of inserting mathematical symbols available in professional text editor. Subsequently, we shall also learn to use macros available with advanced editors to replicate basic code blocks.

The other issue of using a MathML enabled browser is extremely important as mathML rendering would ,otherwise, be severely distorted. Most of the advanced browser has inherent capability in this regard or with available plug-ins. This aspect, however, may not be overlooked for implementation of MathML features in its complete manifestation – particularly when such browsers are available free and takes less than 5 minutes to install on the desktop.

We must understand that MathML specifications are W3C guidelines for the renderer (the product or program that displays content like browsers). It is, therefore, always likely that there are differences in details in the style of presentation by different browsers – even though they are MathML compatible. Notwithstanding the subtle differences, the browsers should be capable to implement MathML codes and hence the requirement of one such to experiment with MathML codes.

Further, we need a separate folder to work with the tutorials and experiment with MathML codes. In the following paragraphs, we enlist precise instructions to prepare the enviornment for MathML code testing :

1: Create a separate working folder for experimenting with the mathML codes (say C:/MathML/testing).

2: In case, you do not have MathML compatible browser already installed on your desktop, download Mozilla Firefox from www.mozilla.org. Install the same by double clicking the Firefox executable setup file. While installing, let the set up program fetches your preferences and other parameters from the browser already installed on the desktop. The set up program shall eventually ask you to whether set Firefox as the default browser on your desktop. For the time being, you may choose to retain the already installed browser as the default one.

2: Ensure that all fonts required for rendering MathML is installed on your computer. In case, you are using Mozilla browser, visit Mozilla web page <http://www.mozilla.org/projects/mathml/fonts/>. The page lists “Fonts to Install”. Click the “font installer” hyper link and download the font installer on your desk top. Once the installer is downloaded, just open the same to install the fonts.

3: Visit www.w3.org/Math/testsuite. It gives you vast array of MathML examples to work with. The website provides codes of hundreds of examples. At certain stage of this course, you may like to copy and paste the codes to your desktop to confirm results. Besides, it is a repository of about 900 entity references used to display symbols and implement mathematical ideas. On the testsuite web page, click the link named "testsuite.zip" to download the same in the desktop folder created for the course. Store the zip file for use in future.

Note: The official documentation of mathML is available at W3C web site www.w3.org/TR/MathML2.

Creating a mathML enabled document

Here, we present the steps to generate MathML codes in MS Word desktop editor. The idea right now is not to explain MathML codes, but to show the way to create a proper document with MathML codes and implement the same in the MathML enabled browser on your desktop.

As the codes need to be free from formatting details and other encoding, the file created is saved as plain text with Unicode(UTF-8) specification. As such, the file should be saved as plain text file with extension “xml” in order to view the implementation in the browser.

Once the document has been created, the same can be tested in MathML supported browser like Mozilla’s Firefox. When MathML codes display the output to the satisfaction, then block of codes starting with <m:math> tag and ending with </m:math> tag can be copied and pasted in the XML document for publication purposes or be saved for future use. Now, proceed with the steps given here to create a mathML enabled “xml” document.

1: Open MS Word. Open a new document. Write exactly or copy/paste the following markup lines in the new document :

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?> <!DOCTYPE math PUBLIC
"-//W3C//DTD MathML 2.0//EN"
"http://www.w3.org/Math/DTD/mathml2/mathml2.dtd" >
<document
xmlns:m="http://www.w3.org/1998/Math/MathML">
</document>
```

2: Write the next top most “xml” element of the document between “document” tags.

```
<content> </content>
```

Note that “document” and “content” are the only XML elements i.e. the pair of tags, used for practicing MathML code. All other code lines, as detailed below, shall be MathML codes.

3: Write “math” MathML element between “content tags” with an example coding to display mathematical construct : $\mathbf{v} = \frac{\mathbf{AB}}{t}$.

4: For the time being, write exactly or copy / paste the code lines as given below and place them between “content” tags :

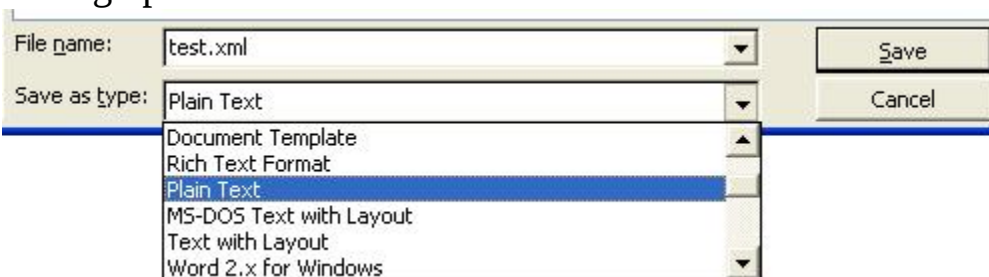
Example:

Displaying mathematical content

```
<content> <m:math display="block"> <m:mrow> <m:mi  
mathvariant="bold">v</m:mi> <m:mo>=</m:mo>  
<m:mfrac> <m:mi mathvariant="bold">AB</m:mi>  
<m:mi>t</m:mi> </m:mfrac> </m:mrow> </m:math>  
</content>
```

5: Click File from Word’s top menu and select “save as”. A dialog box appears. Look at the "Save as type" option. Select “Plain Text” option from the drop down menu. Give a name to the file such as “test.xml”. Ensure that you have provided “xml” extension. Click “Save” button available at the right hand corner (also visible in the figure below).

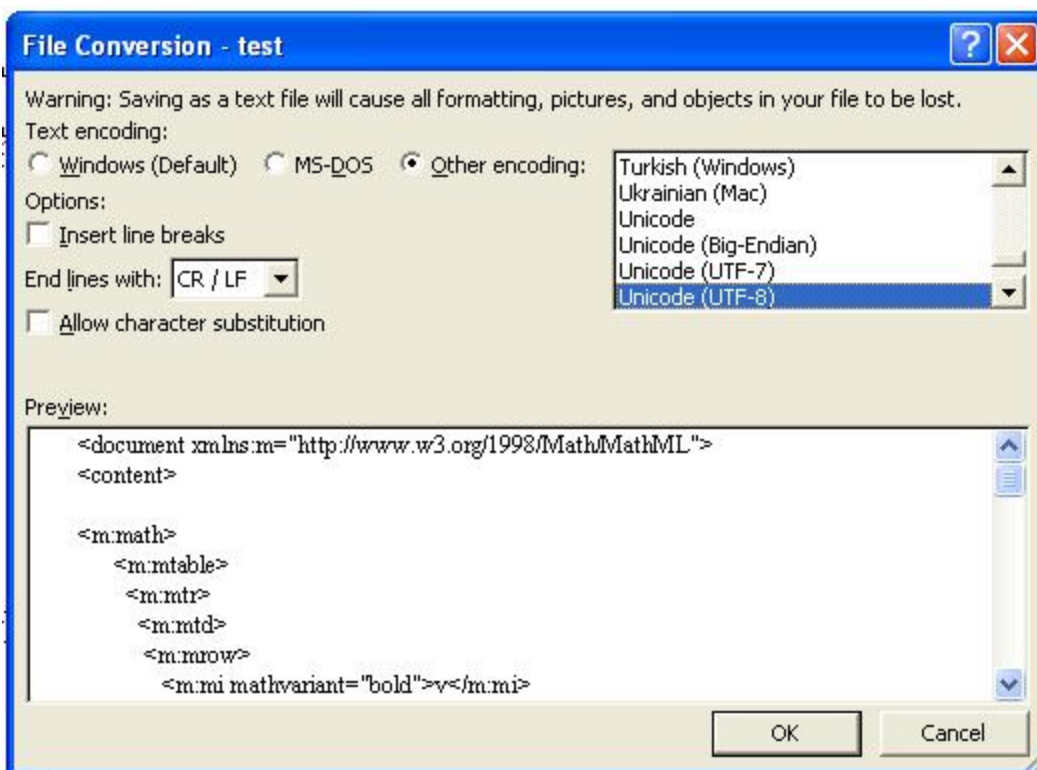
Saving options



Select “plain text” from drop-down and give a name with
xml extension

6: "File Conversion" dialog box appears as shown below. Select the radio button "Other encoding"; Select "Unicode(UTF-8)" option as shown. Click "OK" button. Save the document to the folder you have created for the tutorial.

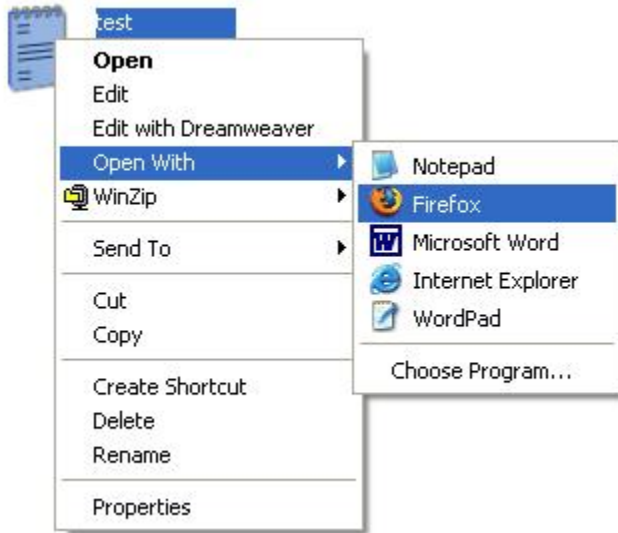
File conversion



Select "other coding" radio button and "Unicode(UTF-8)" option

7: Using "My Computer" or Window Explorer go to the tutorial folder, where you saved the document in earlier step. Select the file and right click the mouse. Select the option "open with".

Opening with mathML enabled document



Select “Open with” from drop-down
and select the mathML compatible
browser

8: If MathML compatible browser is not in the list, select “Choose program” and include the browser that you have downloaded for the purpose. The executable file of the MathML compatible browser must be in the “Program files” folder on your desktop.

9: The mathML code is implemented by the browser as shown below :

$$\mathbf{v} = \frac{\mathbf{AB}}{t}$$

This completes the basic of creating a “xml” document to implement MathML codes. Subsequently, when you are satisfied by the implementation, you may copy the MathML code lines and paste the same in the document to be published.

Context of MathML

Presentation of mathematical content has been a problem for text editors and web renderers, owing largely to the requirement of (i) arrangement of characters and symbols of different sizes (ii) their staggered alignment and (iii) overall structural arrangement different from normal text. As a consequence, MathML markup has been developed to address the needs of the web authors, overcoming these requirements.

The marking up of MathML tags, however, itself is inherently a lengthy and error prone proposition, requiring frequent and repeated testing in order to compose a “well formed” mark up code. A simple notation such as x^{10} may require 6 lines of coding with strict adherence to the syntax and grammar of the mark up language.

Indeed, development of MathML is not targeted for direct use. In the language of Web Consortium (W3C) “..... Instead, it is anticipated that authors will use equation editors, conversion programs, and other specialized software tools to generate MathML. Alternatively, some renderers and systems supporting mathematics may convert other kinds of input directly included in Web pages into MathML on the fly, in response to a cut-and-paste operation.....”.

Though, using tailored software is the easiest way around, but a general understanding of the mathML mark up language may let you grasp the environment of mathematical content publication and at the same time let you be independent of the individual software adaptation issues; while simultaneously letting you continue with developing codes in an organized and efficient manner. If you adapt well to the MathML, there is every chance that you might not use a specialized tool at all.

General features of mathML coding

1: MathML tries to capture the requirement of both presentation and content (meaning) of mathematical expression. Take the example of trigonometric function “sin x”. Simply stated it is an identifier “sin” and a character “x”, which are separated by a white space. From the perspective of mathematics, however, “sin” is a function, which operates on an operand “x”. The two aspects are different when perceived from the angle of presentation and

interpretation. MathML provides coding frame work for both these aspects through two different sets of mark-up called “Presentation” and “Content” mark-ups. As pointed out earlier, we would, however, be limited to the presentation mark-up.

2: MathML elements are preceded with “m:”, specifying the namespace for implementing the codes by the renderers i.e. browsers. A namespace associates a unique global identifier (usually a URL) with a particular set of tags and their usage rules.

3: MathML coding is case sensitive. Mostly, it is lower case coding, unless stated otherwise.

4: MathML markup adheres to the HTML markup structure. Most elements begin with a start tag and end with an end tag. Example :
`</m:mi>a</m:mi>`. Ofcourse, there are also few empty tags.

5: MathML implementation is very touchy about the syntax and structure. A start tag must always be closed by an end tag. There are few varieties in the manner they are closed, but then they all must be closed. Empty elements, for example, are closed in the beginning itself.

6: There is no white space allowed between enclosing angles of a tag. The code parser i.e. the browser would not accept a space inside the angles of the tag like `< mi>`. It should be written without white space as `<mi>`.

Displaying mathematical content

The topmost element in mathML is "math" element.

We need to display mathematics in different situations and in different ways. When we do not specify how to display the content, then “math” element simply displays the mathematical content starting from the left side of the browser. We can, however, specify display by providing values to the “display” attribute of math element. This attribute takes two values : "inline" and "block".

It must be emphasized that "math" element is the top level element and serves as the constructor for all MathML elements. A MathML mark ups without this top level element would result in implementation error.

Inline display

Let us consider the implementation of "math" element with its "display" attribute set to the value "inline". The starting tag of "math" element looks like `<m:math display="inline">` :

Example:

Inline display

```
<m:math display="inline"> <m:mrow> <m:mi
mathvariant="bold">v</m:mi> <m:mo>=</m:mo>
<m:mfrac> <m:mi mathvariant="bold">AB</m:mi>
<m:mi>t</m:mi> </m:mfrac> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\mathbf{v} = \frac{\mathbf{AB}}{t}$$

Now, we edit the above file, making a single change. We specify the start tag of the "math" element as `<m:math>`. We find that the display of the content in the browser with the edit affected is same as when "display" attribute was given "inline" value. This is because, the attribute value

“inline” is actually the default value of “math” element. However, this left hand display in the browser is not the main feature of "inline" specification. More importantly, "inline" attribute allows us to display the math content embedded right into the bulk of text. See the code below to display two subscripted variables (for the time being, do not bother to know the elements used). Just copy the code and paste the same in a new MS Word document and save as “test.xml”. The display in the MathML enabled browser is shown in the example here :

Example:

Inline display

```
<para id="para1" The object is moving with two
different speeds <m:math display="inline">
<m:msubsup> <m:mi>v</m:mi> <m:mn>1</m:mn>
<m:mrow/> </m:msubsup> </m:math> and <m:math
display="inline"> <m:msubsup> <m:mi>v</m:mi>
<m:mn> 2</m:mn> <m:mrow/> </m:msubsup> </m:math>
in two equal time intervals. Find the average
speed. </para>
```

The display looks like :

The object is moving with two different speeds v_1 and v_2 in two equal time intervals. Find the average speed.

Block display

Normally, mathematical content is displayed separated from the body of text as a block. To facilitate this display, we need to provide "block" value to "display" attribute of the "math" element.

Example:

Block display

The velocity is mathematically expressed in terms of displacement and time as :
$$\mathbf{v} = \frac{\mathbf{AB}}{t}$$

Save the file after editing as “test.xml”. Note that the main body of the text is clearly separated from body of the text.

The velocity is mathematically expressed in terms of displacement and time as :

$$\mathbf{v} = \frac{\mathbf{AB}}{t}$$

MathML elements

There are scores of elements in MathML. They are grouped under two broad headings :

1. Presentation elements
2. Content elements

Presentation elements describe visual representation of mathematical notation or content in a two-dimensional plane. In other words, these elements emphasize the visual attributes like size, color, alignment and relative positions. The content elements, on the other hand, emphasize the mathematical “sense” of the expression. There are nearly 30 presentation elements and 50 attributes. Content elements are more in numbers - about 120, but they accept only about a dozen attributes.

Presentation elements

Presentation elements fall under following categories :

1: Token elements : These elements display data and text. We must know that the token elements are the only MathML elements permitted to contain character data. It is relevant to know that MathML character data set is Unicode characters, which are about 900 in numbers. In total, the token elements are seven (7) in numbers. The names and their basic role in rendering content are listed here :

Token elements

- mi : renders identifiers and symbols
- mn : renders numbers
- mo : renders mathematical operators
- mtext : renders text
- mspace : creates space in both horizontal and vertical directions
- ms : renders content for processing by other programs
- mglyph : enables creating new characters

2: Layout elements : They serve as placeholders for text and data at different levels of display matrix or as mechanism to distribute mathematical content in two dimensional space. There are three broad categories of layout elements :

(i) Basic layout elements : They provide basic framework for structuring mathematical expression in two dimensional space. The basic layout elements are ten (10) in numbers. The names and their basic role in rendering layout are listed here :

Basic layout elements

1. “mrow” : makes horizontal groups of sub-expressions for variety of purposes
2. mfrac : renders elements in $\frac{P}{Q}$ form
3. msqrt : renders a square root
4. mroot : renders a radical with an index
5. mstyle : sets style attributes for the enclosed elements
6. merror : enclose a syntax error message from a preprocessor
7. mpadding : manages space around content by manipulating the size of box surrounding content
8. mphantom : makes content invisible, while maintaining structure

9. mfenced : surrounds its content with a pair of fences and segregates content's terms with separators
10. menclose : enclose content with a stretching symbol

(ii) Scripting elements : They provide the mechanism to render sub and super script, under and over script and such other scripting lay out. , under provides For example, msup and msub provide mechanism to script a base character with a pair of scripting characters such as v_2 and A_1 . The

scripting lay out elements are seven (7) in numbers. The names and their basic role in rendering layout are listed here :

Basic layout elements

1. msub : displays subscript
2. msup : displays superscript
3. msubsup : displays sub and super script simultaneously
4. munder : displays under script
5. mover : displays over script
6. munderover : displays under and over script simultaneously
7. mmultiscripts : displays multiple vertically sub and super scripts simultaneously

(iii) Table forming elements : They provide schemata for the formation of a table. They are six (6) in numbers :

Table elements

1. mtable : creates table
2. mlabeledtr : creates a labelled row
3. mtr : creates a row
4. mtd : creates a column
5. maligngroup : Grouping element
6. malignmark : creates alignment markings

Child elements and arguments

Most of the elements (other than empty elements in content MathML) enclose child elements. Some of the elements, however, require that certain numbers of child elements or blocks of child elements be present. In such

cases, these conditional child elements or blocks of child elements are called arguments.

Consider the example. Here “msub” element requires a minimum of two child elements. The child elements “mo” and “mi” in this example are, therefore, called arguments.

Example:**Arguments**

```
<m:math display="block"> <m:msub> <m:mi> A </m:mi>  
<m:mn> 1 </m:mn> </m:msub> </m:math>
```

Save the file after editing as “test1.xml” and see the display. The display looks like :

$$A_1$$

Consider another example given below. Here, “mfrac” element requires exactly two arguments. In case, there are more than two child elements, then “mrow” is to place child elements in two blocks as “mfrac” would take only two arguments –first block representing numerator and second block representing denominator of a ratio. The group of child elements enclosed within each pair of "mrow" tags constitutes one argument of the "mfrac" element.

Example:**Child elements**

```
<m:math display="block"> <m:mrow> <m:mi>v</m:mi>  
<m:mo>=</m:mo> <m:mfrac> <m:mrow> <m:mi>a</m:mi>  
<m:mo>+</m:mo> <m:mi>b</m:mi> </m:mrow>  
<m:mi>c</m:mi> </m:mfrac> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$v = \frac{a + b}{c}$$

Creating table

We have chosen this advance topic right in the beginning with a purpose. Matter of fact, it is important to introduce this topic from the point of view of presenting display of example output in a well structured form. The basic purpose here is to acquire capability to distribute display of math content in different parts of the browser window – without taking recourse to “xml” elements. Hence, our purpose is not to know the details of creating various types of tables in mathML (we shall learn this topic in details later), but to know only elementary idea about creating table so that we get the capability to format the mathematical content display as per the requirement in hand.

For the reason cited as above, the creation of table, is limited to creating invisible table (without border) with rows and columns so that mathematical expressions are suitably placed on the matrix of the browser and is not rendered simply as a continuous line of display.

The top most table element "mtable" contains "mtr" (row) and "mtd" (column) elements. A general construct for 2 rows 2 columns is given here. It is not very difficult to infer from the code lay out that "mtr" element, following "mtable" element defines a single row, which in turn, are split in columns as defined by "mtd" elements.

Example:

Codes for creating table

```
<m:math display="block"> <m:mtable> <m:mtr>
<m:mtd> .... </m:mtd> <m:mtd> .... </m:mtd>
</m:mtr> <m:mtr> <m:mtd> .... </m:mtd> <m:mtd>
.... </m:mtd> </m:mtr> </m:mtable> </m:math>
```

We can now use the table matrix to fill in with the mathematical content. A simple example is given below :

Example:

Structured display

```
<m:math display="block"> <m:mtable> <m:mtr>
<m:mttd> <m:mtext>The value of x is :</m:mtext>
</m:mttd> <m:mttd> <m:mn>10</m:mn> </m:mttd> </m:mtr>
<m:mtr> <m:mttd> <m:mtext>The value of y is :
</m:mtext> </m:mttd> <m:mttd> <m:mn>20</m:mn>
</m:mttd> </m:mtr> </m:mtable> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

The value of x is : 10

The value of y is : 20

Grouping with mrow element

This topic is also a preparatory in the sense that we encounter this element now and then in any of coding effort we undertake. Its primary function within a parent element is to combine child elements enclosed between its tags as one unit for the parent element and align them horizontally. This grouping is important in order to meet the requirement of elements, which takes certain number of arguments.

The MathML coding frame work, however, is aware of the cluttering of codes due to repetitive use of this element. For this reason, "mrow" element is inferred even if not applied with certain elements. It is not necessary to remember the list of these elements capable to infer "mrow" as the same is implied logically. The elements, which require zero or one or exactly one argument, enforce child elements inside it to be clubbed together as one argument. For this reason, elements, requiring zero or one or exactly one

argument, infer “mrow” even without explicit "mrow" tags. A square root element, for example, takes zero or one argument and as such implements “mrow” implicitly. Similar is the case with “mtd” element and other elements, requiring one argument. It must be understood that if there is only one child element, then there is no need to implement "mrow" for grouping either implicitly or explicitly.

Inferred “mrow” is a great help. Imagine if you were to write a pair of “mrow” tags for each of the “mtd” column tags for creating a simple table.

MathML infrastructure

The MathML infrastructure comprises of three important segments of resource base :

MathML resource base

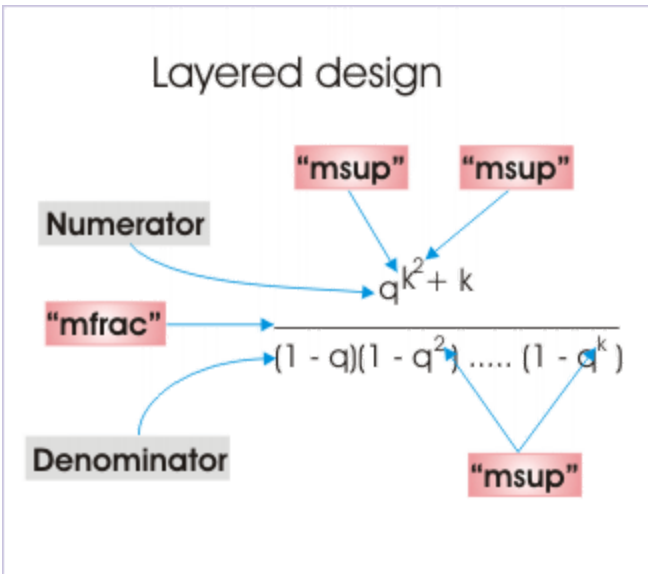
- Elements
- Unicode character set
- Attributes

Elements

The MathML captures markup requirements for mathematical content display through layered encapsulation of markup design, using three groups of elements (token, layout and table elements).

The underlying philosophy of the mark up design is easily understood in terms of the implementation of a fraction in $\frac{P}{Q}$ form. Once the basic form of the fraction is coded, then second layer of coding is taken up for the numerator expression, followed by third layer of coding for the denominator expression. Thus, we see that coding paradigm for a fraction in the form of $\frac{P}{Q}$ is layered over each other with the core of “mfrac” element. It must be emphasized that we refer layering in reference to the sequence of coding requirements, which should be written one after another and not the display, which is structured in two dimensions. The illustration shown in the figure displays a portion of “Rogers-Ramanujan identity” :

Layered coding



Coding is implemented around key layout elements

We see that "mfrac" element provides the core structure for nominator and denominator. Subsequently, either of numerator or denominator or both may have elements, which may further require display in two dimensions. Matter of fact, numerator expression, in the illustration above, consists of nested super scripts.

This type of layered coding design is an immense help in generating code for MathML. In contrast to the two dimensional display requirements, the coding requirement is essentially linear. The display of "Rogers-Ramanujan identity" is a point in the case. The figure below outlines the coding sequence. Note that we proceed writing code from left to right, encapsulating two dimensional aspects along the way, such as subscripting and ratio formation with MathML constructors. And we do it all, one after another in sequence.

Sequential coding

Linear coding from left to right

$$1 + \sum_{k=1}^{\infty} \frac{q^{k^2+k}}{(1-q)(1-q^2)\dots(1-q^k)} = \prod_{j=1}^{\infty} \frac{1}{(1-q^{5j+2})(1-q^{5j+3})}, \text{ For } |q| < 1$$

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫

① mi ② msunderover ③ mfrac ④ mfrac (numerator)
 ⑤ mfrac (denominator) ⑥ mi ⑦ msunderover ⑧ mfrac
 ⑨ mfrac (numerator) ⑩ mfrac (denominator) ⑪ mi ⑫ mi

Coding is implemented in sequence

Unicode character set

The MathML displays data, symbols and text with the help of token elements, which are capable to render a set of certain characters. In all cases of rendering mechanism provided in MathML, the display output is a character or a series of characters. These characters are defined as the Unicode set of characters – about 900 in numbers and growing all the time to meet the requirement of mathematics in conveying special meaning to its expressions. Different font variants of a letter are treated as separate symbols as they may represent different mathematical meaning/ concept/ idea.

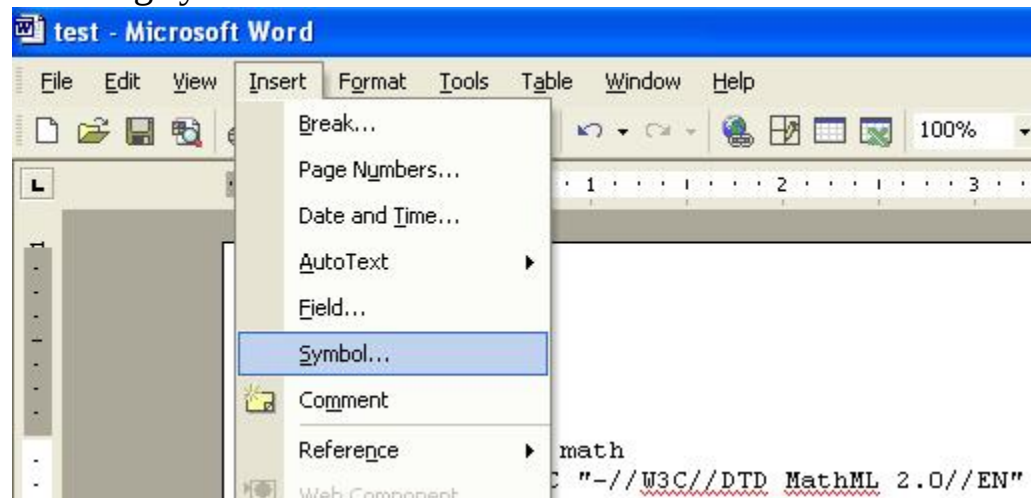
There are three ways to encode Unicode characters with token elements specialized in displaying characters:

- 1: Use the keyboard and type the characters as available on the keyboard.
- 2: Use the name reference known as “entity reference”. Encode the entity reference preceded by "&" and ending with semicolon “;”. For example, we would encode entity reference “beta” as "β" to represent “β”.
- 3: Use the corresponding Unicode numeric hexadecimal reference. For example, beta is encoded with its hexadecimal numeric reference "U003B2" as "&x003B2;" to represent “β”.

It is evident that keyboard typing is the easiest option. The difficulty is that keyboard supports only a limited numbers of characters. We can type ASCII characters holding “Alt” key and typing numbers from the numeric key pad on the right side of the keyboard. However, this approach is not very helpful as we have no clue of the visual form of the character corresponding to a number.

The best approach for encoding Unicode characters is to create a XML file with extension "xml" in the desktop Word program as instructed earlier in the course. Then, select top level “Insert” menu of the MS Word program and choose “Symbol”.

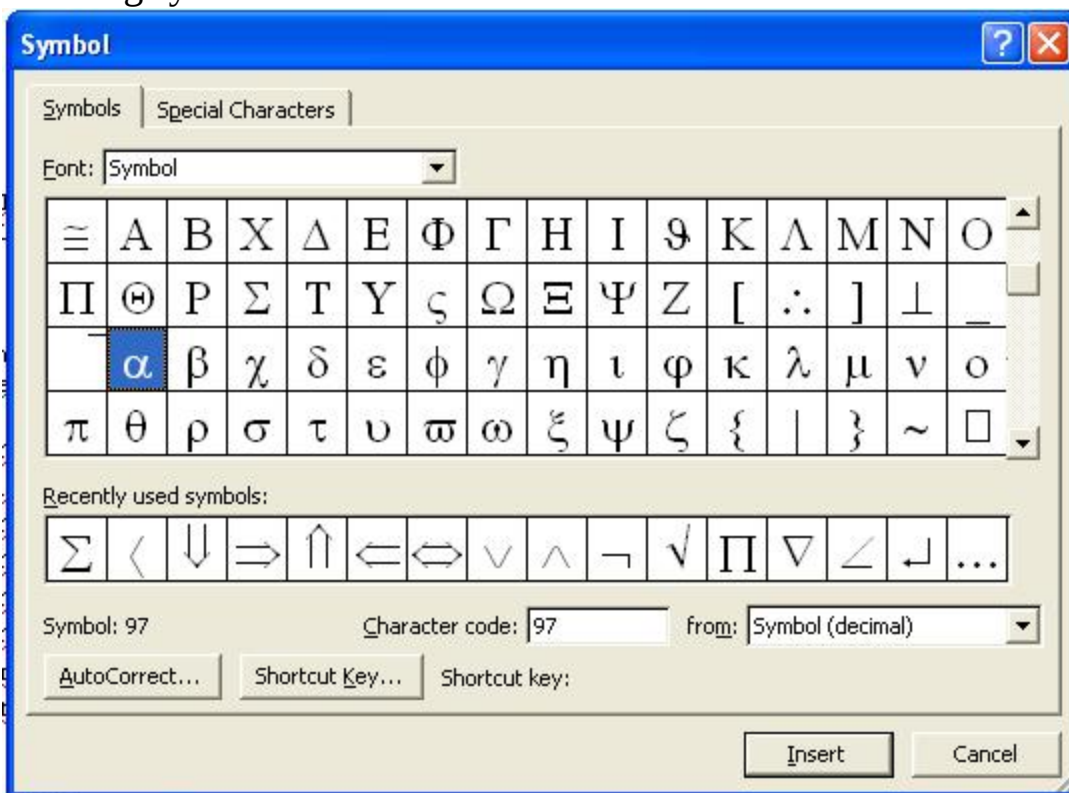
Inserting symbols



A “symbol” dialog box appears as shown below. It is important that we select the option “normal text” from the “Font” drop down list. Keep selecting and inserting the characters not available on the keyboard, but useful for our application as we foresee. Save the "xml" file under some

meaningful name like "unicode characters" and reference the special characters as when required from the file.

Inserting symbols



This technique serves a great part of requirement, but for the characters which do not have visual forms like “ThickSpace”, “ApplyFunction”, “nbsp” etc., which are called ”invisible” characters. But these invisible characters are few in numbers. As such, we can store their entity reference names for easy reference as and when required.

The second approach of using entity reference like "β" is the second best approach as the names suggest the visual forms for many important characters. The third alternative to use Unicode numeric reference in the form "&x003B2;" is not desirable for those who intend to code MathML. Ofcourse, this could be the best approach for software developers – probably a numeric representation would be easier to handle in the programming algorithm.

A list of entity reference along with its display form is presented in the Appendix. The listing is divided in three modules ([A to H](#)) , ([I to Q](#)) and ([R](#)

[to Z](#)) to accommodate about 900 Unicode characters. You may choose to visit these pages to have a first hand experience of what these characters look like and how they are named.

The example given here demonstrates character rendering, using each of the encoding techniques discussed above.

Example:

Display using Unicode hexadecimal number

```
<m:math display="block"> <m:mtext>Alpha is  
displayed as : </m:mtext> <m:mi> α</m:mi> <m:mi>  
&alpha; </m:mi> <m:mi> &#x003B1;</m:mi> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

Alpha is displayed as : $\alpha\alpha\alpha$

Some characters falling under “operator” category, beside dpsce rendering characters, do not have visual display. They are called non-marking characters and are used with “mo” element. These characters are important for the quality of print or alternative rendering like audio rendering. They provide specific and consistent spacing on the operand on which they act, thus distinguishing the mathematical operation from normal character rendering. The non-marking operator are :

```
"&InvisibleTime;", "&InvisibleComma;" and  
"&ApplyFunction;"
```

These non-marking characters must be distinguished from space or blank rendering characters, which are used to format display for improving readability like " ", " ", "&emsp4;", " " etc. In addition to these space rendering elements, “mspace” token element is handy, where we need to implement and manage space.

Managing attributes

MathML elements, including token elements, support an array of attributes. Its management is greatly simplified in a structured manner. There are four attribute classes and a specialized tool to manage attributes of the elements in MathML. It must, however, be emphasized these are not exclusive groups, but only they represent groups having certain common implementation features.

Attribute class

1. Attributes common to all MathML elements
2. Attributes inherited from the rendering environment
3. Attributes common to a group of token elements
4. Attributes specific to an element (additional attributes)

All MathML elements supports “class”, “style”, “id”, “xlink:href” and “xref” attributes in order to use style sheet mechanism. If the renderer does not use a style sheet, then these attributes may simply be ignored.

The style attributes of MathML elements are inherited from the rendering environment. We can change these inherited attributes from the environment in two ways. First mechanism is provided by common attribute design for token elements. Second mechanism involves using “mstyle” element belonging to layout presentation category.

Attributes common to a group of token elements

Rendering by token elements is characterized by the default values used for attributes of these elements. Most of these attributes are inherited from the environment of renderer like a particular browser.

The style attributes of the individual element can, however, be set different to default values via a set of style attributes. To keep the matter simple, there is a group of attributes, which applies to five of the token elements all capable to render characters (“mi”, “mn”, “mo”, “mtext”, “ms”) and one layout element “mstyle”.

The common attribute mechanism is a great help in managing and studying style attributes for the content display. This feature allows us to concentrate on specific attributes of the element while common attributes are managed with common names for six elements.

The common style attribute values are :

Attribute values types

- **mathvariant** (default =normal for all, italic for “mi”) : normal | bold | italic | bold-italic | double-struck | bold-fraktur | script | bold-script | fraktur | sans-serif | bold-sans-serif | sans-serif-italic | sans-serif-bold-italic | monospace
- **mathsize** (default = inherited) : small | normal | big | number vertical unit
- **mathcolor** (default = inherited) : #rgb | #rrggbb | html-color-name
- **mathbackground** (default = inherited) : #rgb | #rrggbb | html-color-name

The example below sets most of these attribute values and is suggestive of the ways attributes are set, using common attributes.

Example:

Using common attributes

```
<m:math display="block"> <m:mrow> <m:mi
mathvariant="bold" mathsize="2em"
mathcolor="Blue"> A </m:mi> <m:mo> + </m:mo> <m:mi
mathvariant="italic" mathsize="10pt" > A </m:mi>
<m:mo> + </m:mo> <m:mi mathvariant="bold-italic">
A </m:mi> <m:mo> + </m:mo> <m:mi
mathvariant="double-struck"> A </m:mi> <m:mo> +
</m:mo> <m:mi mathvariant="bold-fraktur"
mathbackground="red"> A </m:mi> <m:mo> + </m:mo>
<m:mi mathvariant="script"> A </m:mi> <m:mo> +
</m:mo> <m:mi mathvariant="fraktur" mathsize="1cm"
mathcolor="1cm"> A </m:mi> <m:mo> + </m:mo> <m:mi
```

```
mathvariant="sans-serif" mathsize=".2in" > A  
</m:mi> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$\textcolor{blue}{A} + A + \textbf{A} + \mathbb{A} + \textcolor{red}{21} + \mathcal{A} + \mathfrak{A}$

MathML also enables use of deprecated mechanism of setting common attributes. The deprecated attributes are :

Attribute values types

- Fontsize : number v-unit
- fontstyle(normal | italic)
- fontweight (normal | bold)
- color : #rgb | #rrggbb | html-color-name
- fontfamily : string

However, if both mechanisms are used to set a particular attribute, then new mathstyle attribute prevails over deprecated mechanism.

Specialized style setting tool

The only layout element “mstyle” in the common attribute group is a single point specialized tool to set varieties of style attributes to elements enclosed by “mstyle” element. This is kind of one go attribute setting mechanism, which can set not only the common attributes, but additional attributes inherited from the rendering environment. Further, “mstyle” element can be used to set style attributes on any MathML elements in addition to the five (5) token elements, which implement common token attributes. In this sense, this element serves as an extremely powerful tool to manage style attributes in MathML. Let us consider setting attributes to token elements as shown here.

Example:

Using common attributes

```
<m:math display="block"> <m:mstyle  
mathvariant="bold" color="blue" mathsize="1cm">  
<m:mi> A </m:mi> <m:mo> + </m:mo> <m:mi> B </m:mi>  
</m:mstyle> </m:math>
```

Save the file after editing as “test.xml”. The “ ” display looks like :

$$\mathbf{A} + \mathbf{B}$$

Managing values

A lot of coding efficiency in mark up language like MathML depends on assigning appropriate values to the attributes. Usually, there exists more than one ways to assign attribute's value.

Size attributes may be expressed either in terms of predefined numeric values as “small”, “normal” or “big” or as numerical values. There are variety of units available to express numerical values. These units are used on the basis of context, some of which are expressed in relative terms with respect to the normal size : “em” or “ex” or “%”. Others are absolute values like “pt” (for point; 1 point = 1/72 inch), “px” (for pixel), “pc” (for picas; 1 pica = 12 points), “in” (for inch), “mm” (for millimeter) and “cm” (for centimeter).

In some cases, number can be assigned to an attribute. The number indicates the relative size with respect to default implementation. The “mfrac” element for example implements “linethickness” attribute. A number, say, “2” indicated that the bar will be drawn twice in thickness with respect to the default thickness as determined by a particular renderer.

In some cases, like with “mo” element, we can assign terms of predetermined space to space controlling attributes like “lspace” and “rspace”. These terms are called “named space”. The named space used in

MathML may be one of "veryverythinmathspace", "verythinmathspace", "thinmathspace", "mediummathspace", "thickmathspace", "verythickmathspace", or "veryverythickmathspace".

The mathcolor or color is set in three possible ways :

1: rgb scheme : Color is considered to be composed of the components red, green and blue. The component colors may be mixed in various proportions. The components are represented by single hexadecimal digit (0,1,...,9,a,b,c,d,e,f). This number is then preceded by “#” and assigned as : mathcolor =”#00f”.

2: rrggbb scheme : This is a similar to the rgb scheme except that each of the component is represented by two digit hexadecimal number. Thus, smallest pair is “00” and largest pair is “ff”. This number is then preceded by “#” and assigned as : color =”#0000ff”.

3: html-color-name : We can assign mathcolor or color attribute with HTML color names : "aqua", "black", "blue", "fuchsia", "gray", "green", "lime", "maroon", "navy", "olive", "purple", "red", "silver", "teal", "white", and "yellow".

The MathML provides control on individual levels of layout structure elements like "table" element. The control requires setting attributes on individual rows and columns. In such cases, attributes are provided with values, which are read in sequence and applied on consecutive rows and columns as the case may be.

Presentation token elements (mi,mn)

There are seven (7) token elements designed to serve the purpose of rendering characters to (i) display (“mi”, ”mn”, ”mo” and ”mtext”) mathematical content (ii) render characters for interfacing with other programs (“ms”) (iii) manage space around characters (mspace) and (iv) create new characters (mlyph).

The presentation token elements are the only elements to take and display characters in MathML. Mathematical symbols or identifiers are either single character (Example : x) or multiple characters (Example : sin). In mathematics, simple rendering of text is not sufficient. Mathematical content uses large numbers of symbols with varieties of visual forms and styles. In normal text rendering, font style does not matter, but different font styles convey different mathematical ideas and concepts and as such are treated as different characters. Besides, there are convention in mathematics about spacing around operators and relative size that needs to be negotiated with unequal size of other characters. To capture the complete spectrum of mathematical details, MathML employs about 900 characters and a specialized system of constructors that render mathematical content in consistent and coherent manner to match the precise nature of mathematical symbols and expressions.

It is interesting to know that rendering of mathematical content by token elements - even if typed directly as string of characters and looking similar on browser display - is inherently different to simple text rendering in a horizontal sequence. The difference is visible when seen in the context of uses that require scripting like sub or super scripts, stretching of operators (like a parenthesis is required to grow to cover the height of expression enclosed etc) etc.

In MathML, the differentiation operator “ⅆ” , for example, could have been represented by letter “d” and hence it may be argued why to include a separate character to represent the symbol of differentiation, which could be easily rendered using letter “d” from keyboard. The reason lies in the style convention of mathematics that requires this operator to have a specific style and form different to a normal character “d”.

Furthermore, these token constructors provide many automatic behavior to character representation consistent with the convention in mathematics. An operator like “+” sign can be rendered with or without space depending upon the context in which the sign appears in the expression.

See the output of the MathML codes in the example given below. Inspect the space before and after the sign “+” in the output, the first “+” sign is very close to the identifier “a”, while the second “+” is surrounded with greater amount of space in horizontal direction. These detailed display design consideration is extremely important to reflect the preciseness required in displaying mathematics.

Example:

Context based spacing in MathML

```
<m:math display="block"> <m:mrow> <m:mo> + </m:mo>  
<m:mi> x </m:mi> <m:mo> + </m:mo> <m:mi> y </m:mi>  
</m:mrow> </m:math>
```

Save the file after editing as “test3.xml”. The display looks like :

$$+x + y$$

The point of specialized token construct is visually made through yet another example given below. Two simultaneous examples are implemented, using an entity reference “→”. Note the impact of using “mover” element on the character reference “→”. Compare the two outputs. Though the lay out of the arrow and text has not changed, but the arrow is stretched to accommodate the length of the text !

Example:

Modified rendering

```
<m:math display="block"> <m:mrow> <m:mi> x </m:mi>  
<m:mo> &RightArrow; </m:mo> <m:mtext> maps to
```

```

</m:text> <m:mi> y </m:mi> </m:mrow> </m:math>
<m:math display="block"> <m:mrow> <m:mi> x </m:mi>
<m:mover> <m:mo> &RightArrow; </m:mo> <m:mtext>
maps to </m:mtext> </m:mover> <m:mi> y </m:mi>
</m:mrow> </m:math>

```

Save the file after editing as “test.xml”. The display looks like :

$$x \rightarrow \text{maps to } y$$

"mi" token element

The “mi” element is used to display mathematical identifiers. Identifiers in MathML include literals, constants, name of the angle, trigonometric function etc. The example, here, displays both single and multi character identifiers.

Example:

Display with mi element

```

<m:math display="block"> <m:mtext>I is displayed as
: </m:mtext> <m:mi> I </m:mi>
<m:mtext>&nbsp;&nbsp;&nbsp;and trigonometric sine
function is displayed as : </m:mtext> <m:mi> sin
</m:mi> <m:mi> θ </m:mi> </m:math>

```

Save the file after editing as “test3.xml”. The display looks like :

I is displayed as :*I*and trigonometric sine function is displayed as : sin θ

Here " " has been used to create two spaces to improve readability.

Note that default rendering of a single character by "mi" element is italic – as against "normal". Multiple characters are, however, displayed as normal. It means that if we want to have a normal display, then we would require to override the default behavior of the ‘mi’ element and specify normal display, using “mathvariant” attribute as demonstrated in the example below.

Example:

Display with mi element

```
<m:math display="block"> <m:mtext>Normal I is  
:&nbsp;&nbsp;&nbsp;</m:mtext> <m:mi mathvariant='normal'>  
I </m:mi> </m:math>
```

Save the file after editing as “test3.xml”. The display looks like :

Normal I is :I

The presentation style of a single character in italic form is not without any reason. This suggests that we might be using single character mostly in italic form for mathematical display. In any case, if you differ, you can change the same to normal.

The “mi” element renders character or sequence of characters without any surrounding space. This element together with other character rendering elements (“mn” and “mo”) forms the building blocks of mathematical display, using MathML. It must also be observed that “mi” implements common attributes available for the token element and does not implement additional attributes or attributes specific to the element.

"mn" token element

The “mn” element is a flexible constructor to display unsigned real numbers of various types and formats, including decimals or combination of characters, which convey real number. In the scheme of the rendering of mathematical content under MathML environment, it is preferred to keep sign

as an operator and its rendering with "mo" element. This approach is consistent with spacing (inter elements) requirement of operator display.

Example:

Display of real numbers

```
<m:math display="block"> <m:mtext>Forty: </m:mtext>
<m:mn> Forty ;</m:mn> <m:mtext>10 : </m:mtext>
<m:mn> 10 ; &nbsp;</m:mn> <m:mtext>4.23: </m:mtext>
<m:mn> 4.23 ; &nbsp;</m:mn> <m:mtext>1,000:
</m:mtext> <m:mn> 1,000 ; &nbsp;</m:mn>
<m:mtext>0.1e10: </m:mtext> <m:mn> 0.1e10 ; &nbsp;</m:mn>
<m:mtext>CX: </m:mtext> <m:mn> CX ; &nbsp;</m:mn>
</m:math>
```

Save the file after editing as “test.xml”. The display looks like :

Forty:Forty ;10 :10 ;4.23:4.23 ;1,000:1,000 ;0.1e10:0.1e10 ;CX:CX ;

Note that default rendering style of the characters are “normal” and as such, we do not need to use “mathvariant” attribute to set the display normal. The “mn” element, however, is not suitable for all kinds of numbers. For example, following constants are displayed using “mi” element : π $\Im$ E

Example:

Numbers displayed by mi element

```
<m:math display="block"> <m:mtext>Complex number :
&ThinSpace;</m:mtext> <m:mn> 1 </m:mn> <m:mo> +
</m:mo> <m:mn> 2 </m:mn> <m:mo> &InvisibleTimes;
</m:mo> <m:mi> &ImaginaryI;</m:mi> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

Complex number : $1 + 2i$

A ratio, on the other hand, is presented with the help of “mfrac” element. See the example code for displaying a ratio of the form (p/q) :

Example:

Numbers not displayed by “mfrac” element

```
<m:math display="block"> <m:mtext>Ratio is :  
</m:mtext> <m:mfrac> <m:mn> 22 </m:mn> <m:mn> 7  
</m:mn> </m:mfrac><m:mi> ; </m:mi> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\text{Ratio is : } \frac{22}{7}$$

Presentation token element (mo)

Specific attributes : form fence separator lspace rspace stretchy symmetric
maxsize minsize largeop movablelimits accent

The “mo” element displays an operator or other representations, which are treated as an operator in mathematics. The term "operator" also includes fence, separator, accent, comma, semicolon, invisible characters etc. , some of which are used to provide new meaning to the ordinary operator. Hence, “operator” in MathML has wider meaning beyond ordinary operators, consistent with the requirement of growing expanse of mathematical operations.

Many of the important operators contained within “mo” element can be typed directly. They are available on key board; while many others have to be referenced through valid Unicode entity references. The operator symbols, which can be typed from the keyboard include “+”, “-”, “/”, “*”, “(”, “)”, “{”, “}”, “[”, “]”, “.NOT.”, “.OR.” etc.

Following example shows the display of operators by "mo" element.

Example:

Displaying operators with “mo” element

```
<m:math display="block"> <m:mtable> <m:mtr> <m:mtd> <m:mi>Plus :  
</m:mi><m:mo>+</m:mo> <m:mi> ;</m:mi> <m:mi>Increment : </m:mi>  
<m:mo>++</m:mo> <m:mi> ;</m:mi> <m:mi>Logical not : </m:mi>  
<m:mo>.NOT.</m:mo> <m:mi> ;</m:mi> </m:mtd> </m:mtr> <m:mtr>  
<m:mtd> <m:mi>Less than and equal : </m:mi><m:mo> &le; </m:mo>  
<m:mi> ;</m:mi> <m:mi>Partial operator : </m:mi><m:mo> &PartialD;  
</m:mo> <m:mi> ;</m:mi> <m:mi>Differentiation : </m:mi><m:mo>  
&DifferentialD; </m:mo> <m:mi> ;</m:mi> <m:mi>Integration :  
</m:mi><m:mo> &int; </m:mo> </m:mtd> </m:mtr> </m:mtable>  
</m:math>
```

Save the file after editing as “test.xml”. The display looks like :

Plus : + ;Increment : ++ ;Logical not : .NOT. ;
Less than and equal : $\leq$;Partial operator : ∂ ;Differentiation : d ;Integration : $\int$

In the above example, we have used “mtable” (with “mtr” and “mtd” elements) element for controlling display. See that first three operators in above example have been typed directly from the keyboard; while others have been referenced by entity reference.

MathML design of rendering an operator closely follows mathematical convention. It distinguishes between rendering a character with “mi” element and an operator with “mo” operator in many important ways for notational representation of mathematical expression. Two basic considerations are (i) to manage space around the operator, consistent with the context in which they are displayed and (ii) dimensional change (stretching) of operators in accordance with the dimension of other

elements and terms. Space around an operator is managed by “form” attribute, while stretching is managed by “stretchy” attribute of “mo” element. A plus operator (“+”) may, for example, precede or follow an identifier or may lie between two operands. Depending upon its placement, the space around the operator “+” is determined. Similarly, size of a parentheses around an expression must stretch to the height of the expression as in $\left(\frac{a}{b}\right)$.

MathML also allows improvisation of an operator rendered by “mo” element. This generally makes use of other characters rendering elements such as “mi” or “mn” elements in addition to “mo” element and their combination with layout elements like “mfrac” and scripting elements (“msub”, “msup” etc.) Simply put, MathML treats an expression with a “mo” element at core and improvised by other elements - as an operator of class known as “embellished” operator.

Consider encoding for a differentiation operator “ⅆ”. This operator when applied to a variable, say t, is represented as $\frac{d}{dt}$. In mathematics, however, $\frac{d}{dt}$ is itself considered to be an operator. The whole block consisting of “mfrac” element is, thus, an embellished operator.

```
<m:mfrac> <m:mo> &DifferentialD; </m:mo> <m:mrow> <m:mo>
&DifferentialD; </m:mo> <m:mi> x </m:mi> </m:mrow> </m:mfrac>
```

Additional attribute : Default values

Default values to the attributes of “mo” element are set in accordance with the values contained in “operator dictionary”, which specifies “form”, “fence”, “stretchy”, “lspace” and “rspace” attributes for different operators. The renderer maintains an operator dictionary for most of the operators. W3C recommends a prototype of operator dictionary. These recommendations can be viewed at www.w3.org/TR/MathML2/appendixf.html. If the dictionary does not provide the value for the attribute, then attribute is set with default value as given here:

Default values

1. form : set by position/context
2. fence : false
3. separator : false
4. lspace : thickmathspace
5. rspace : thickmathspace
6. stretchy : false
7. symmetric : true
8. maxsize : infinity
9. minsize : 1
10. largeop : false
11. movablelimits : false
12. accent : false

Managing space around operators

Attribute value types

- form : prefix | infix | postfix
- lspace : number h-unit | namedspace
- rspace : number h-unit | namedspace

The "form" attribute specifies whether an operator shall be rendered as "prefix" or "postfix" or "infix" operator. These form types determine space around the operator (left and right of it). Each form type is associated with two attributes ("lspace" and "rspace"), which implements its left space ("lspace") and right space ("rspace"). It is important to understand that form types are basically specification about space around operator and about its role with respect to other elements.

The default value of "form" attribute is set in accordance with three rules, involving "mrow" element. The token elements, including "mo" element, is composed within explicit or inferred "mrow" element. The position of an operator within the "mrow" in horizontal sequence determines the form type of the operator. If operator is the first element, then its form is inferred as "prefix"; if it is in the intermediate position, then its form is inferred as "infix"; and if it is in the end position, then its form is inferred as "postfix". Corresponding to each form, operator library specifies "lspace" and "rspace". It is, however, not necessary that each of the operator has all the three form types. The nature of operator determines the type of forms that a particular operator should be associated with. For example, "+" operator has entries for "prefix" and "infix", but not for "postfix" in the operator dictionary, because "postfix" form of "+" operator is not expected to be placed at the end of an expression.

Now, consider the example of "+" operator appearing twice in an expression in the manner as typed below :

+a + b

The first "+" sign in the above case assumes "prefix" form, while the second "+" sign assumes "infix" form. Once, the form of the operator is determined from the context within "mrow" element, spacing around the operator is determined as specified by "lspace" and "rspace" values given in the operator dictionary. The dictionary suggests following values for these forms of "+" operator :

"+" form="infix" lspace="mediummathspace" rspace="mediummathspace"
"+" form="prefix" lspace="0em" rspace="veryverythinmathspace"

Evidently, "infix" form separates its neighboring elements by "mediummathspace", whereas "prefix" form separates the following element with a very small space specified by "veryverythinmathspace".

Let us now code the example in MathML and see the output :

Example:

Determining "form" of an operator

```
<m:math display="block"> <m:mrow> <m:mo> + </m:mo> <m:mi> a  
</m:mi> <m:mo> + </m:mo> <m:mi> b </m:mi> </m:mrow> </m:math>
```

Save the file after editing as "test.xml". The display looks like :

+a + b

As expected first “+” is almost attached to “a”, where as second instance of “+” renders some amount of space around itself. This type of detailing in rendering mathematical expression is vital and critical to maintain highest order of rendering consistency, which is commensurate with the exact nature of mathematics and the meaning that a mathematical expression conveys. Note that the mark-up paradigm of MathML automatically forces codification in such a manner that no extra effort is required towards maintaining conventions of mathematical display - almost ruling out the possibilities that an expression is displayed in a non-conforming way.

A particular operator does not require to have entries for all the forms in the operator dictionary. Many of the operators in mathematics operate on other entities in forward direction. For example, integration, differentiation, partial differentiation etc. operates on identifiers following it. For this reason, such operators are inherently “prefix” in nature. On the other hand, operators classified as fences are either “prefix” or “postfix” commensurate with their role in mathematical expression. It is very unlikely that a closing bracket “]” is used in the beginning of an expression and as such it has “postfix” form. Dictionary entries for these operators are given here to understand : why there is very small space or no space between these operators and the identifiers on which they operate.

```
&DifferentialD;" form="prefix" lspace="0em"
rspace="verythinmathspace" "&PartialD;" form="prefix" lspace="0em"
rspace="verythinmathspace" "&Integral;" form="prefix"
largeop="true" stretchy="true" lspace="0em" "(" form="prefix"
fence="true" stretchy="true" lspace="0em" rspace="0em" ")"
form="postfix" fence="true" stretchy="true" lspace="0em"
rspace="0em" "[" form="prefix" fence="true" stretchy="true"
lspace="0em" rspace="0em" "]" form="postfix" fence="true"
stretchy="true" lspace="0em" rspace="0em" "{" form="prefix"
fence="true" stretchy="true" lspace="0em" rspace="0em" "}"
form="postfix" fence="true" stretchy="true" lspace="0em"
rspace="0em"
```

We are at liberty to specify these attributes. It is, however, recommended that we leave the arrangement to the system, which is rendering the mathematical content. It shall ensure consistency in the display, which follows the convention of mathematics as implemented by a particular renderer. The example below demonstrates how we can change the spacing different to default and against the form values as inferred from the context :

Example:

Determining "form" of an operator

```
<m:math display="block"> <m:mrow> <m:mo form="infix"
rspace="10pt"> + </m:mo> <m:mi> a </m:mi> <m:mo form="prefix"> +
</m:mo> <m:mi> b </m:mi> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$+ \quad a + b$$

Stretching of operators

Four attributes of “mo” element control stretching of operators. These attributes may assume following values :

Attribute value types

- stretchy : true | false
- symmetric : true | false
- maxsize : number [v-unit | h-unit | namedspace | infinity
- minsize : number [v-unit | h-unit | namedspace

Among the operators, the requirement for stretching of fences, arrows, accents (angular cap on identifier) and separators are most profound and visible in mathematical expressions. For this reason, “stretchy” attribute of fence and accent operators are set “true” in operator dictionary. Stretching of operators in an expression is restricted by “minsize” and “maxsize” attributes.

The stretchable operators are characterized as predominantly either vertically or horizontally stretchable. The fences, various kinds of vertical arrows (single or double), operators like $\prod$, $\sum$, $\int$, “/” etc. are set to stretch vertically by default in operator directory.

When stretchable operator and non-stretchable terms are bounded by explicit or inferred “mrow” element, then the stretchable operator grows vertically to cover the non-stretchy term. Consider the example given here :

Example:

Vertical stretching

```
<m:math display="block"> <m:mrow> <m:mi> x </m:mi> <m:mo> =  
</m:mo> <m:mo> ( </m:mo> <m:mfrac> <m:mi>a</m:mi> <m:mi>b</m:mi>  
</m:mfrac> <m:mo> ) </m:mo> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = \left(\frac{a}{b} \right)$$

We can, however, control the growth of parentheses by setting “maxsize” attribute to 1 i.e. equal to its normal size.

Example:

Vertical stretching

```
<m:math display="block"> <m:mrow> <m:mi> x </m:mi> <m:mo> =  
</m:mo> <m:mo maxsize="1"> ( </m:mo> <m:mfrac> <m:mi>a</m:mi>  
<m:mi>b</m:mi> </m:mfrac> <m:mo maxsize="1"> ) </m:mo> </m:mrow>  
</m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = \left(\frac{a}{b}\right)$$

Thus, setting "maxsize" attribute overrides the default behavior, which allows the parentheses to stretch and cover the non-stretchy expression. Let us, now experiment with other than fence character like $\sum$ and observe their behavior with other terms :

Example:

Vertical stretching

```
<m:math display="block"> <m:mrow> <m:mo> &sum; </m:mo> <m:mo> (
</m:mo> <m:mfrac> <m:mi>A</m:mi> <m:mi>B</m:mi> </m:mfrac> <m:mo>
) </m:mo> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\sum\left(\frac{A}{B}\right)$$

Control for stretching can be selective as well. For example, we can set stretchy=”false” on the opening parenthesis to restrict it to grow.

Example:

Vertical stretching

```
<m:math display="block"> <m:mrow> <m:mi> x </m:mi> <m:mo> =
</m:mo> <m:mo stretchy="false"> ( </m:mo> <m:mfrac> <m:mi>a</m:mi>
<m:mi>b</m:mi> </m:mfrac> <m:mo> ) </m:mo> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = \left(\frac{a}{b}\right)$$

In situation where, the expression bounded by “mrow” tags contains terms of different heights, the stretchable parentheses grow to cover the highest of the terms.

Example:

Vertical stretching

```
<m:math display="block"> <m:mrow> <m:mi> x </m:mi> <m:mo> =
</m:mo> <m:mo> ( </m:mo> <m:mfrac> <m:mi>a</m:mi> <m:mi>b</m:mi>
</m:mfrac> <m:mo> + </m:mo> <m:mfrac> <m:mrow> <m:mfrac>
<m:mi>c</m:mi> <m:mi>d</m:mi> </m:mfrac> </m:mrow> <m:mi>e</m:mi>
</m:mfrac> <m:mo> ) </m:mo> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = \left(\frac{a}{b} + \frac{\frac{c}{d}}{e} \right)$$

The symmetric attribute is designed to stretch operator in both vertical and horizontal direction from the axis of the characters in equal magnitude. The symmetric attribute applies only to characters, which can stretch vertically; otherwise this attribute is ignored. Usually, this attribute is set “true” for vertically stretchable operator, but in certain cases involving matrix of unequal size, we may prefer to set it “false” as demonstrated in the example here :

Example:

Vertical stretching

```
<m:math display="block"> <m:mrow> <m:mi> x </m:mi> <m:mo> =
</m:mo> <m:mo symmetric="false"> ( </m:mo> <m:table
align="bottom"> <m:mtr> <m:td> <m:mi>a</m:mi> </m:td> <m:td>
<m:mi>b</m:mi> </m:td> </m:mtr> <m:mtr> <m:td> <m:mn>d</m:mn>
</m:td> <m:td> <m:mn>e</m:mn> </m:td> </m:mtr> </m:table>
<m:mo symmetric="false"> ) </m:mo> <m:mo symmetric="false"> (
</m:mo> <m:table align="bottom"> <m:mtr> <m:td> <m:mi>a</m:mi>
</m:td> <m:td> <m:mi>b</m:mi> </m:td> </m:mtr> <m:mtr> <m:td>
<m:mn>d</m:mn> </m:td> <m:td> <m:mn>e</m:mn> </m:td> </m:mtr>
<m:mtr> <m:td> <m:mn>d</m:mn> </m:td> <m:td> <m:mn>e</m:mn>
</m:td> </m:mtr> </m:table> <m:mo symmetric="false"> ) </m:mo>
</m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = \begin{array}{cc} & a & b \\ a & b & d & e \\ d & e & d & e \end{array}$$

Matrix operation of unequal sizes uses notation which is aligned to base. This type of controlling stretching in specific direction is, therefore, extremely useful in such situations.

In case, the “mrow” domain contains non-stretchable terms of normal height and other stretchable terms, then all the terms, including the fence operator grows to the maximum normal height.

Example:**Vertical stretching**

```
<m:math display="block"> <m:mrow> <m:mi> x </m:mi> <m:mo> =
</m:mo> <m:mo> ( </m:mo> <m:mo> &int; </m:mo> <m:mi> f</m:mi>
<m:mo> ( </m:mo> <m:mi> x </m:mi> <m:mo> ) </m:mo> <m:mi>
&DifferentialD; </m:mi> <m:mi> x </m:mi> <m:mo> ) </m:mo>
</m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = \left(\int f(x) dx \right)$$

As against fence operator, the accent and horizontal arrows are stretchable in horizontal direction by default. The growth, in this case, is controlled by “munder”, “mover” and “munderover” elements, which contain the operator. The “munder”, “mover” and “munderover” elements, as the names suggest, allow drawing of an operator "under" or "over" or both about a character(s) or expression. The scripting elements takes a base argument about which the operator is to be drawn and one (“munder” and “mover”) operator or two (“munderover”) operators for being placed about the base. We shall see that the operator grows horizontally to cover the other element.

Example:**Horizontal stretching**

```
<m:math display="block"> <m:mi> A </m:mi> <m:munderover> <m:mo>
&RightArrow; </m:mo> <m:mtext> 50 degree C </m:mtext> <m:mtext>
200 psi </m:mtext> </m:munderover> <m:mi> B </m:mi> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$A \begin{matrix} 200 \text{ psi} \\ 50 \text{ degree C} \end{matrix} B$$

Other attributes : largeop, movablelimits, accent, separator, fence

These attributes may assume following values :

Attribute value types

- fence : true | false
- separator : true | false
- largeop : true | false
- movablelimits : true | false
- accent : true | false

All these attributes accept boolean values “true” or “false”. The attribute “fence” is designed for non-visual rendering like audio rendering. As such, this attribute has no impact on the visual aspect of rendering. The role of “separator” attribute is also not significant and may be left to default value.

The “largeop” attribute determines the size of the operator. If it is true, then the operator is drawn larger than its normal size. For example, $\int$ and $\prod$ operators are displayed as large operator as “largeop” attribute for these operators are set “true” by default. If we need to display the normal size, then their “largeop” attribute is set to “false”.

Example:

Large operator

```
<m:math display="block"> <m:mrow> <m:mo> &prod; </m:mo> <m:mo> , </m:mo> <m:mo> &int; </m:mo> <m:mo> , </m:mo> <m:mo largeop="false"> &prod; </m:mo> <m:mo> , </m:mo> <m:mo largeop="false"> &int; </m:mo> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\prod, \int, \prod, \int$$

The movablelimits attribute allows underscripts and overscripts to be represented as subscript and superscript respectively. This rendering of under and over scripts as sub and super scripts is possible by setting this attribute to true. The implementation of this attribute by renderers is not yet consistent.

Example:

The “movablelimits” attribute

```
<m:math display="block"> <m:munderover> <m:mo movablelimits='true'> &sum; </m:mo> <m:mi> a </m:mi> <m:mi> b </m:mi> </m:munderover> <m:munderover> <m:mo movablelimits='false'> &sum; </m:mo> <m:mi> a </m:mi> <m:mi> b </m:mi> </m:munderover> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\sum_a^b \sum_a^b$$

The accent attribute determines whether an operator is treated as accent (diacritical mark) when used as an underscript or overscript.

Presentation token elements (others)

"mtext" token element

The “mtext” element plays similar role of rendering characters like “mo” and “mn” elements, but with a difference. It does not convey the notional or symbolic aspect of mathematical identities and numbers, which require specific presentation and manipulation to convey mathematical meaning. The “mtext” element is used to present plain text, which may be used for comments and clarification.

This element is also useful to deliver white space and mixing verbose mathematical rendering consisting of word string. For example, we would need to say “The presentation of alpha character in MathML looks like : α ”.

Example:

Display with "mtext" element

```
<m:math display="block"> <m:mtext>The presentation  
of alpha character in MathML is : </m:mtext>  
<m:mi>&alpha;</m:mi> </m:math>
```

Save the file after editing as “test4.xml”. The display looks like :

The presentation of alpha character in MathML is : α

Though “mtext” element manages the empty space within the string of words, but not around it. In order to enhance, readability, style and presentation, we may add empty space, using "&ThinSpace" entity character.

Example:

Display with "mtext" element

```
<m:math display="block"> <m:mtext>The presentation  
of alpha character in MathML is : &ThinSpace;  
</m:mtext> <m:mi>&alpha;</m:mi> </m:math>
```

Save the file after editing as “test4.xml”. The display looks like :

The presentation of alpha character in MathML is : α

Color and size management of the content of “mtext” element is implemented with the common attributes of the token element i.e. mathvariant, mathsize, mathcolor and mathbackground directly on "mtext" element.

Example:

Display with "mtext" element

```
<m:math display="block"> <m:mtext  
mathvariant='bold' mathcolor='#fff'  
mathsize='12pt' mathbackground='#f55'> Example :  
</m:mtext> <m:mtext mathsize='10pt'> The above  
example illustrates application of “mfrac”  
element. </m:mtext> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

Example : This example illustrates application of theorem.

In general, mixing of text with mathematics may be implemented, using any of the three possibilities discussed here. First, we can rely on the rendering facility of the document in which “MathML” is embedded. This document could be any valid “xml” document, including HTML. Second, we may use the text rendering by “mtext” element as explained above. This is appropriate for text, which is neither the part of the general text flow in the

document nor the part of the notional display of mathematical expression. It suits the requirement of text rendering, used for qualifying mathematical expression or comments. Third, displaying text with “mi” element, which is inline with the mathematical expression or a part of it.

"mspace" token element

Attribute value types

- width (default : "0em"): number h-unit | namedspace
- height (default : "0ex"): number v-unit
- depth (default : "0ex"): number v-unit
- linebreak (default : "auto"): auto | newline | indentingnewline | nobreak | goodbreak | badbreak

This element is provided to render a blank space of any size, which can be controlled with the help of attribute values. It is a more flexible mechanism in comparison to the space rendering characters like " ", "&emsp", "&emsp14".

The “mspace” element has three attributes to control the size of the blank space (width, height and depth). Significantly, its default values are set such that its rendering does not affect normal presentation. It means that if we want to affect the layout, we would have to assign attribute values for the size. Size attribute accept numbers in variety of ways, using different units.

The “linebreak” attribute, on the other hand, suggests different line breaking options for the renderer to implement.

Consider the example given here,

Example:

Using mspace element to manage space

```
<m:math display="block"> <m:mfrac> <m:mrow>  
<m:mi>a</m:mi> <m:mspace width="0.5in"/>  
<m:mi>b</m:mi> </m:mrow> <m:mrow> <m:mi>c</m:mi>
```

```
<m:mspace height="1.27cm"/> <m:mi>d</m:mi>  
</m:mrow> </m:mfrac> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\frac{a}{b}$$

Besides, there is an important issue with regard to uses of “mspace” and also other elements, which introduce space on rendering. Space rendering has no visual lymphs or character display, but despite this, space rendering elements are counted as far as counting of arguments are concerned. The element “mfrac”, which is used to display ratio, takes only two arguments – one of which forms the numerator and other forms the denominator of the ratio. In the example below, we simply introduce “mspace” between the “mfrac” tags. The MathML parser (renderer) counts “mspace” and as such fails to implement code as there are three arguments as against the requirement of exactly two arguments for “mfrac” element. As expected, the browser notifies that the code is "invalid-markup".

Example:

Counting of "mspace" as an argument

```
<m:math display="block"> <m:mfrac> <m:mi>a</m:mi>  
<m:mspace height="0.5in"/> <m:mi>b</m:mi>  
</m:mfrac> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

mfrac

Thus, in order to meet the requirement of elements with fixed arguments, we would have to group child elements with the help of “mrow” element.

```
<m:mfrac> <m:mi>a</m:mi> <m:mrow> <m:mspace  
height="0.5in"/> <m:mi>b</m:mi> </m:mrow>  
</m:mfrac>
```

"ms" token element

Specific attributes : lquote rquote

Attribute value types

- lquote (default : “”) : string
- rquote (default : “”) : string
- rspace : number h-unit | namedspace

This is also a character rendering element, but with a double quote around. This is provided for interpretation of MathML data by other programs and may not be very useful for the authors or for the requirements limited to display.

By default, both the left and right quote characters are standard double quote character ,". We can change the surrounding characters with the lquote and rquote attributes respectively.

Let us now work with the example given here to view the output for few cases.

Example:

Display with "ms" element

```
<m:math display="block"> <m:mrow> <m:ms>A</m:ms>  
<m:mo>+</m:mo> <m:ms lquote='|'  
rquote='|'|>A</m:ms> <m:mo>+</m:mo> <m:ms>&#x26;  
</m:ms> <m:mo>+</m:mo> <m:ms>Hello, I have come to  
meet you.</m:ms> <m:mo>+</m:mo>
```

```
<m:ms>&DifferentialD;</m:ms> <m:mo>+</m:mo> <m:ms  
lquote='a' rquote='b'>&int;</m:ms> </m:mrow>  
</m:math>
```

Save the file after editing as “test4.xml”. The display looks like :

“A” + |A| + “&” + “Hello, I have come to meet you.” + “*d*” + a**f**b

"mglyph" token element

The "mglyph" element is intended for creating new character in addition to what is available with Unicode character set. With the expanding horizon of mathematics, it is possible that a character with new "glyph" is required to represent new mathematical idea or concept. On the working level, the "mglyph" element draws on the existing Unicode character or some "font style", which must be supported by the renderer. For our course, though, we feel that this topic may not be very useful. As such, this element is not discussed in any more detail here.

Presentation layout elements (mrow, mfrac, msqrt, mroot)

Layout elements control two dimensional layout of characters, symbols and composed expressions. There are three groups of elements that determine the two dimensional layout of the mathematical content and affects its management.

Layout element groups

- Basic layout elements
- Scripting elements
- Table elements

Basic layout elements comprises of 10 elements. The names and their basic role in rendering layout are listed here :

Basic layout elements

1. “mrow” : makes horizontal groups of sub-expressions for variety of purposes
2. “mfrac” : renders elements in P/Q form
3. “msqrt” : renders a square root
4. “mroot” : renders a radical with an index
5. “mstyle” : set environment attributes for the enclosed elements
6. “merror” : enclose a syntax error message from a preprocessor
7. “m padded” : manages space around content
8. “mphantom” : makes content invisible, while maintaining structure
9. “mfenced” : Segregates terms with separators and surround its content with a pair of fences
10. “menclose” : enclose content with a stretching symbol

”mrow” layout element

The “mrow” element is a horizontal sub-grouping facility. Grouping of sub expression is an important requirement for variety of reasons. Primarily, this element determines the relationship between an “operator” and “operand”. When there is single character “operator” and “operand” relationship, then grouping is not an issue. However, when there more than one “operand” in an expression, then grouping is required.

Grouping of elements is also important where elements require certain numbers of argument. In such situation, it is possible that there are more child elements than the arguments required. Consider the example here, where child elements are grouped to meet the requirement of “mfrac” syntax for two arguments.

Example:

”mfrac” layout element

```
<m:math display="block"> <m:mfrac> <m:mrow> <m:mi>x</m:mi>  
<m:mo>+</m:mo> <m:mn>y</m:mn> </m:mrow> <m:mn>z</m:mn> </m:mfrac>  
</m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\frac{x + y}{z}$$

In the example above, “mrow” element is used to group the sub-expression “x + y” to be treated as numerator in “mfrac” implementation.

We have also seen earlier that “mrow” plays important role in determining the form of an operator. The form of an operator as “prefix” or “infix” or “postfix” is determined by the position of an operator in the group of sub expression defined by “mrow” element.

Grouping of elements, terms and sub-expressions (in yet another sub-expression) is a common requirement for meaningful mathematical expression. As such, MathML platform implements an inferred mechanism, whereby certain elements infer that the child elements are grouped with “mrow” in an implicit manner. This means that we may save on writing a pair of “mrow” tags in such situation.

The elements, which implements inferred “mrow” are : “msqrt”, “mstyle”, “merror”, “menclose”, “mpadded”, “mphantom”, “mtd”, and “math”. Important to note here is the inclusion of top level “math” element in the list. It means that putting “mrow” tag just after “math” element is actually redundant.

If we look closely at the elements, which implement inferred “mrow”, then we observe that all these elements require to treat everything inside it as a single unit. This is the reason that they would otherwise require a pair of “mrow” tags to combine all child elements as one group. It is, therefore, good thinking on the part of MathML developers to enforce “mrow” inexplicitly and save on coding and avoid cluttering with “mrow” tags.

”mfrac” layout element

Specific attributes : linethickness numalign denomalign bevelled symmetric

The “mfrac” element renders fraction in $\frac{p}{q}$ form. The element requires two arguments to implement the fraction. The first argument is numerator, while the second is denominator of the fraction. The syntax of the element is as given below :

```
<mfrac> numerator denominator </mfrac>
```

The “mfrac” element is an extremely flexible element to produce very complicated fraction, in which numerator or denominator may themselves be a fraction. Such implementation is affected by nested “mfrac” as shown in the example here.

Example:
Nested “mfrac” display

```
<m:math display="block"> <m:mi> x </m:mi> <m:mo> = </m:mo>
<m:mfrac> <m:mrow> <m:mn> 1 </m:mn> <m:mo> + </m:mo> <m:mfrac>
<m:mn> 3 </m:mn> <m:mn> 10 </m:mn> </m:mfrac> </m:mrow> <m:mrow>
<m:mfrac> <m:mn> 2 </m:mn> <m:mn> 7 </m:mn> </m:mfrac> <m:mo> +
</m:mo> <m:mfrac> <m:mn> 12 </m:mn> <m:mn> 5 </m:mn> </m:mfrac>
</m:mrow> </m:mfrac> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = \frac{1 + \frac{3}{10}}{\frac{2}{7} + \frac{12}{5}}$$

In the example, the numerator of the outermost “mfrac” element contains expression consisting of another “mfrac” element, while the denominator consists of two “mfrac” elements. Also note the use of “mrow” element to ensure that a “mfrac” elements takes exactly two arguments.

Changing bar thickness

Attribute value types

- linethickness (default = 1) : number | thin | medium | thick

The fraction is displayed with a fraction bar or rule, separating numerator and denominator. Appearance of this bar can be changed using “linethickness” attribute. The default thickness of the bar is set by the rendering environment. When set to zero, no bar is shown.

A thicker bar, in relation to default thickness, can be shown assigning values to “linethickness” attribute. MathML provides predetermined thin, medium and thick values, which is a preferred technique to display different thicknesses of the bar. This way, we can maintain uniformity in our display. Normally, bar thickness is not required to be changed except when nested “mfrac” is implemented. In that case, there are more than one bar and as such it may be required to distinguish the central bar from others. The example given here encapsulates these aspects of “mfrac” display. .

Example:

Distinguishing line thickness in "mfrac" display

```
<m:math display="block"> <m:mi> x </m:mi> <m:mo> = </m:mo>
<m:mfrac linethickness="thick"> <m:mrow> <m:mn> 1 </m:mn> <m:mo>
+ </m:mo> <m:mfrac linethickness="medium"> <m:mn> 3 </m:mn>
<m:mn> 10 </m:mn> </m:mfrac> </m:mrow> <m:mrow> <m:mfrac> <m:mn>
2 </m:mn> <m:mn> 7 </m:mn> </m:mfrac> <m:mo> + </m:mo> <m:mfrac>
<m:mrow> <m:mfrac linethickness="medium"> <m:mn> 8 </m:mn> <m:mn>
```

```
11 </m:mn> </m:mfrac> </m:mrow> <m:mn> 5 </m:mn> </m:mfrac>
</m:mrow> </m:mfrac> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = \frac{1 + \frac{3}{10}}{\frac{2}{7} + \frac{\frac{8}{11}}{5}}$$

Bevelled fraction and the alignment of rows

Attribute value types

- numalign (default = center) : left | center | right
- denomalign (default = center) : left | center | right
- bevelled (default = false) : true | false

The “numalign” and “denomalign” affects alignment of numerator and denominator with respect to the bar. In most of the case, we do not change the default “center” element as this is the manner in which expressions are written in mathematics.

The “bevelled” attribute provides an alternate form for rendering fraction. In beveled form, the bar is a slant line, in stead of a horizontal line. This form is useful, when we want to include fraction inside the text as its default form takes more vertical space than that of the text, disturbing line spacing between text lines. More significantly, this reduces vertical space of an intensively nested expression.

Example:

Bevelled "mfrac" display

```
<m:math display="block"> <m:mi> x </m:mi> <m:mo> = </m:mo>
<m:mfrac bevelled="true" linethickness="thick"> <m:mrow> <m:mn> 1
</m:mn> <m:mo> + </m:mo> <m:mfrac linethickness="medium"> <m:mn>
3 </m:mn> <m:mn> 10 </m:mn> </m:mfrac> </m:mrow> <m:mrow>
<m:mfrac> <m:mn> 2 </m:mn> <m:mn> 7 </m:mn> </m:mfrac> <m:mo> +
</m:mo> <m:mfrac> <m:mrow> <m:mfrac bevelled="true"
linethickness="medium"> <m:mn> 8 </m:mn> <m:mn> 11 </m:mn>
</m:mfrac> </m:mrow> <m:mn> 5 </m:mn> </m:mfrac> </m:mrow>
</m:mfrac> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x = 1 + \frac{3}{10} \Big/ \frac{2}{7} + \frac{\frac{8}{11}}{5}$$

Radical layout elements

The “msqrt” and “mroot” are radical elements. The “msqrt” element renders square root, while “mroot” element represents indices with a base and index.

"msqrt" layout element

The “msqrt” element accepts zero or one argument. Its syntax is :

```
<msqrt> base </msqrt>
```

When there are more than one child elements, then they are considered to be enclosed within “mrow” tags. This means that the “msqrt” element draws an inferred implementation of “mrow”. This saves us from coding “mrow” tags explicitly, irrespective of numbers of arguments.

Example:

Square root element : msqrt

```
<m:math display="block"> <m:msqrt> <m:mi>x</m:mi> <m:mo>-</m:mo>  
<m:mn>3</m:mn> </m:msqrt> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\sqrt{x - 3}$$

Like other elements in MathML, “msqrt” element supports nesting to enclose square root recursively.

Example:

Nested square

```
<m:math display="block"> <m:msqrt> <m:mi>{</m:mi> <m:mi>x</m:mi>  
<m:mi>+</m:mi> <m:msqrt> <m:mi>(</m:mi> <m:mi>x</m:mi> <m:mi>+  
</m:mi> <m:mi>10</m:mi> <m:mi>)</m:mi> </m:msqrt> <m:mi>}</m:mi>  
</m:msqrt> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\sqrt{\{x + \sqrt{(x + 10)}\}}$$

"mroot" layout element

The “mroot” element requires exactly two arguments – the first one as the base and second as index. The syntax of the element is :

```
<m:mroot> base index </m:mroot>
```

Each of these arguments may be combination of elements, which must be grouped separately with the help of “mrow” element in two distinct groups.

Example:

Indices element : mroot

```
<m:math display="block"> <m:mroot> <m:mrow> <m:mi>x</m:mi>  
<m:mo>-</m:mo> <m:mn>3</m:mn> </m:mrow> <m:mfrac> <m:mi>x</m:mi>  
<m:mn>2</m:mn> </m:mfrac> </m:mroot> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\sqrt[x]{x-3}$$

Presentation scripting elements

Mathematical notation requires scripting around a base entity. The MathML provides scripting capability through scripting elements : `msub`, `msup` and `msubsup`. Scripting in MathML is not limited only to putting a subscript or superscript around an expression, but vertically integrated upper and lower scripts, which are handy to display limits on integrals and to meet such other requirements of rendering structured display.

Rendering of script is done by manipulating certain attributes that elements acquire from the environment such as “xml” or “html” document.

"msub" and "msup" layout scripting elements

Specific attributes : `superscriptshift`, `superscriptshift`

Attribute value types

- `superscriptshift`, `superscriptshift` (default : automatic) : number v-unit

Each of these scripting elements takes two arguments base and subscript, which may be valid MathML expressions. Syntax of the two elements are as given here :

```
<msub> base subscript </msub> <msup> base  
superscript </msup>
```

Each of these scripting elements takes two arguments base and subscript, which may be valid MathML expressions. Implementation of “msub” element shifts the baseline of the script, while retaining that of the base. In the case of “msup” element, the baseline shifts the script up. In both cases, display style of the scripts are also changed

The amount of shifting can be specified by giving numerical values to “subscriptshift” and “superscriptshift” attributes preferably in "ex" unit. The example here demonstrates implementation of both “msup” and “msub” elements.

Example:

Subscript and supersubscript

```
<m:math display="block"> <m:table> <m:mtr> <m:mtd>
<m:msup> <m:mi> m </m:mi> <m:mrow> <m:mn> 1
</m:mn> <m:mo> &InvisibleComma; </m:mo> <m:mn> 2
</m:mn> </m:mrow> </m:msup> </m:mtd> </m:mtr>
<m:mtr> <m:mtd> <m:msub subscriptshift="1.5ex">
<m:mi> m </m:mi> <m:mrow> <m:mn> 1 </m:mn> <m:mo>
&InvisibleComma; </m:mo> <m:mn> 2 </m:mn>
</m:mrow> </m:msub> </m:mtd> </m:mtr> </m:table>
</m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$m^{12}$$

$$m_{12}$$

Observe that “⁣” entity reference impart the meaning that scripts is read one two and not as one multiplied by 2.

"msubsup" layout scripting element

Specific attributes : subscriptshift superscriptshift

Attribute value types

- subscriptshift (default : automatic) : number v-unit
- superscriptshift (default : automatic) : number v-unit

The element takes three arguments. First argument of the element is base; second argument is subscript; and third argument is superscript. The syntax of the element is :

```
<msubsup> base subscript superscript </msubsup>
```

Scripting “msubsup” element affects simultaneous implementation of a subscript and superscript, which are vertically aligned. This is an extremely useful scripting element as this element can be used to display limit values on variety of expression, including an integral sign.

The amount of shifting can be specified by giving numerical values to “subscriptshift” and “superscriptshift” attributes preferably in “ex” unit. The code below demonstrates assigning limiting values to integral with “msubsup” element, while “msup” element is used to construct exponential function.

Example:

Definite integral limits

```
<m:math display="block"> <m:math display="block">
<m:mrow> <m:mi> y </m:mi> <m:mo> = </m:mo>
<m:mrow> <m:msubsup> <m:mo> &int; </m:mo> <m:mn> 0
</m:mn> <m:mn> 1 </m:mn> </m:msubsup> <m:mrow>
<m:msup> <m:mi> &ExponentialE; </m:mi> <m:mi> x
</m:mi> </m:msup> <m:mo> &InvisibleTimes; </m:mo>
<m:mrow> <m:mo> &DifferentialD; </m:mo> <m:mi> x
</m:mi> </m:mrow> </m:mrow> </m:mrow> </m:mrow>
</m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$y = \int_0^1 e^x dx$$

"munder" layout scripting element

Specific attributes : accentunder

Attribute value types

- `accentunder` (default : automatic) : true | false

The “munder” takes two arguments. First one is the base and second one is the scripting character. The syntax of the element is :

```
<munder> base underscore </munder>
```

The scripting character is placed below the base character/ expression. The “accentunder” attribute is either “true” or “false”, which determines whether the underscore is implemented as accent or limit. The implementation of accent is drawn closer to the base and is of the same size as that of base. On the other hand, limit is drawn relative away from the base and is reduced in size.

If under-script is an “mo” element, the value of its “accent” attribute is used as the default value of “accentunder”. However, an explicitly given value overrides the default.

Example:

Underscripting

```
<m:math display="block"> <m:math display="block">
<m:mrow> <m:munder accentunder="true"> <m:mrow>
<m:mi> x </m:mi> <m:mo> + </m:mo> <m:mi> y </m:mi>
<m:mo> + </m:mo> <m:mi> z </m:mi> </m:mrow> <m:mo
stretchy='true'> &OverBar; </m:mo> </m:munder>
<m:mtext> &ThinSpace; vs &ThinSpace; </m:mtext>
<m:munder accentunder="false"> <m:mrow> <m:mi> x
</m:mi> <m:mo> + </m:mo> <m:mi> y </m:mi> <m:mo> +
</m:mo> <m:mi> z </m:mi> </m:mrow> <m:mo
stretchy='true'> &OverBar; </m:mo> </m:munder>
</m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x + y + z \text{ vs } x + y + z$$

"mover" layout scripting element

Specific attributes : accent

Attribute value types

- accent (default : automatic) : true | false

The “mover” takes two arguments. First one is the base and second one is the scripting character. The scripting character is placed below the base character/expression. The syntax of the element is :

```
<munder> base overscript </munder>
```

The “accent” attribute is either “true” or “false”, which determines whether the underscript is implemented as accent or limit. The implementation of accent is drawn closer to the base and is of the same size as that of base. On the other hand, limit is drawn relative away from the base and is reduced in size.

Example:

Overscripting

```
<m:math display="block"> <m:mrow> <m:mover  
accent="true"> <m:mi> x </m:mi> <m:mo> &Hat;  
</m:mo> </m:mover> <m:mtext>  
&ensp;&ensp;vs&ensp;&ensp; </m:mtext> <m:mover  
accent="false"> <m:mi> x </m:mi> <m:mo> &Hat;  
</m:mo> </m:mover> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\hat{x} \quad \text{vs} \quad \hat{x}$$

If over-script is an “mo” element, the value of its “accent” attribute is used as the default value of “accent”. However, an explicitly given value

overrides the default.

Example:

Overscripting

```
<m:math display="block"> <m:mrow> <m:mover  
accent="true"> <m:mrow> <m:mi> x </m:mi> <m:mo> +  
</m:mo> <m:mi> y </m:mi> <m:mo> + </m:mo> <m:mi> z  
</m:mi> </m:mrow> <m:mo stretchy='true'> &OverBar;  
</m:mo> </m:mover> <m:mtext> &ThinSpace; vs  
&ThinSpace; </m:mtext> <m:mover accent="false">  
<m:mrow> <m:mi> x </m:mi> <m:mo> + </m:mo> <m:mi>  
y </m:mi> <m:mo> + </m:mo> <m:mi> z </m:mi>  
</m:mrow> <m:mo stretchy='true'> &OverBar; </m:mo>  
</m:mover> </m:mrow> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$x + y + z \text{ vs } x + y + z$$

"munderover" layout scripting element

Specific attributes : accent accentunder

Attribute value types

- accent (default : automatic) : true | false
- accentunder (default : automatic) : true | false

The “mover” takes three arguments. First one is the base and second and third ones are the under and over scripting characters respectively. The scripting character is placed below and above the base character/expression. The syntax of the element is :

```
<munderover> base underscript overscript  
</munderover>
```

This element allows simultaneous under and over scripting and is tailor-made constructor to render both scripts simultaneously. The scripting with this element suits in providing limit values to integral sign.

Example:

Simultaneous under and over scripting

```
<m:math display="block"> <m:munderover> <m:mo>  
&int; </m:mo> <m:mn> 0 </m:mn> <m:mi> &infin;  
</m:mi> </m:munderover> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\int_0^{\infty}$$

This element is not a necessity though, as the same can be implemented using “munder” and “mover” one after another. The reason of this equivalence is that implementing either “munder” or “mover” does not change horizontal width of the base, which is being scripted.

Example:

Simultaneous under and over scripting

```
<m:math display="block"> <m:mover> <m:munder>  
<m:mo> &int; </m:mo> <m:mn> 0 </m:mn> </m:munder>  
<m:mi> &infin; </m:mi> </m:mover> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\int_0^{\infty}$$

"mmultiscripts" layout scripting element

Specific attributes : subscriptshift superscriptshift

Attribute value types

- subscriptshift (default : automatic) : number v-unit
- superscriptshift (default : automatic) : number v-unit

This element is same as “msubsup” element in that it renders vertically aligned pair of sub and super scripts, but with one enhancement that it can render multiple pairs such scripts. The syntax of the element is as given below :

```
<mmultiscripts> base (subscript superscript)* [  
<mprescripts/> (presubscript presuperscript)* ]  
</mmultiscripts>
```

The multiple sign after pair of subscript and superscript indicates that the pair can be repeated. If it is required to render pair of scripts before the base, then the same is indicated by presence of an empty element “mprescripts” as shown in the syntax. An absence of this empty element will indicate that prescripts are not to be rendered. The pair prescripts are optional as such notation are rare in mathematics. A code for three pairs of the scripts are given in the example :

Example:

Implementation of multiple script pairs

```
<m:math display="block"> <m:mmultiscripts> <m:mi>  
A </m:mi> <m:mi> i </m:mi> <m:mi> j </m:mi> <m:mi>  
k </m:mi> <m:mi> l </m:mi> <m:mi> m </m:mi> <m:mi>  
n </m:mi> </m:mmultiscripts> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$A_{i\,k\,m}^{j\,l\,n}$$

It is evident that the implementation of “mmultiscripts” with one pair of scripts would be equivalent to implementation of “msubsup” implementation. An implementation that requires that one of the script in any pair is missing, we may provide either an empty element <none/> or empty “mi”.

Example:

Implementation of multiple script pairs with missing value

```
<m:math display="block"> <m:mmultiscripts> <m:mi>
A </m:mi> <m:mi> i </m:mi> <m:mi> j </m:mi> <m:mi>
k </m:mi> <m:mi> l </m:mi> <m:mi> </m:mi>
<m:mfrac> <m:mi> x </m:mi> <m:mi> y </m:mi>
</m:mfrac> <m:mprescripts/> <m:mi> a </m:mi>
<m:mi> b </m:mi> <m:mi> c </m:mi> <m:mi> d </m:mi>
</m:mmultiscripts> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$bd\,A_{ac\,i\,k}^{j\,l\,\frac{x}{y}}$$

Presentation table elements

Table element is applied for variety of rendering requirements, in addition to its primary use for drawing a table. The applications of table frame work include displaying two dimensional mathematical content like arrays and matrix, creation of labeled equations and enabling display of content in multiple rows equivalent to line break.

Consider the task to write two rows of equations on two consecutive lines :

$$3x + y = 6$$

$$2x - y = 5$$

How would we force two lines to be displayed in a single MathML constructor? We may probably display the lines as separate rows of a table without a frame and lines. In this sense, table frame work provides a facility, which is equivalent to forcing a new line.

Similarly, we may be required to number important equations so that the same can be referenced subsequently like :

$$3x + y = 6 \dots\dots\dots 1.1$$

In this context, the label attribute available for “mlabeledtr” element allows us to number the equation as above. A matrix, on the other hand, is implemented by a table construct without border and enclosed parentheses :

$$\begin{pmatrix} x & y & z \\ 1 & 2 & 3 \end{pmatrix}$$

The MathML employs six elements to render all sorts of table. However, there is a serious mis-match between specifications of some of these elements and implementation of the same by browsers. For this reason, we shall concentrate on three table forming elements namely "mtable", "mtr" and "mtd" elements, which together form the basic constructors for creating tables (only references to other elements has been given for other three elements).

Table elements

1. mtable : creates table
2. mlabeledtr : creates a labelled row
3. mtr : creates a row
4. mtd : creates a column
5. maligngroup : Grouping element
6. malignmark : creates alignment markings

“mtable” element

Specific attributes : align rowalign columnalign groupalign alignmentscope columnwidth
width rowspacing columnspacing rowlines columnlines frame
framespacing equalrows equalcolumns displaystyle side minlabelspc

The ‘mtable’ element represents a table of rows and columns. It accepts only “mtr” or “mlabeledtr” elements as its child. Both of these define a single row in the table. The only difference between the two row elements is that “mlabeledtr” can be used to label the particular row defined by it. The “mtr” element represents a simple row. Each of these row forming elements takes child elements to define the columns.

The “mtable” element requires that all rows have same numbers of columns. If the columns in any row mismatches, then that row is padded with empty column(s). For example, when one column definition in the first row involving “y” is removed, then tables is rendered as shown below :

Example:**Table short of one column**

```
<m:math display="block"> <m:mtable frame="solid" columnlines="solid"
rowlines="solid"> <m:mtr> <m:mttd> <m:mi>x</m:mi> </m:mttd> <m:mttd>
<m:mi>z</m:mi> </m:mttd> </m:mtr> <m:mtr> <m:mttd> <m:mn>1</m:mn> </m:mttd>
<m:mttd> <m:mn>2</m:mn> </m:mttd> <m:mttd> <m:mn>3</m:mn> </m:mttd> </m:mtr>
</m:mtable> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

x	z	
1	2	3

Evidently, output is not what was intended. It is, therefore, always advisable to maintain same numbers of columns in each row. If there is no content, then that column should be kept as an empty column. Introducing the column definition back without any content restores the structure of table.

```
<m:mttd> </m:mttd>
```

The out put now is :

Example:**Table with same numbers of columns**

```
<m:math display="block"> <m:mtable frame="solid" columnlines="solid"
rowlines="solid"> <m:mtr> <m:mttd> <m:mi>x</m:mi> </m:mttd> <m:mttd> </m:mttd>
<m:mttd> <m:mi>z</m:mi> </m:mttd> </m:mtr> <m:mtr> <m:mttd> <m:mn>1</m:mn>
</m:mttd> <m:mttd> <m:mn>2</m:mn> </m:mttd> <m:mttd> <m:mn>3</m:mn> </m:mttd>
</m:mtr> </m:mtable> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

x		z
1	2	3

Displaying table**Attribute value types**

- align (default : "axis") : (top | bottom | center | baseline | axis) [rownumber]
- rowalign (default : "baseline") : (top | bottom | center | baseline | axis)+
- columnalign (default : "center") : (left | center | right) +
- groupalign (default : "left") : group-alignment-list-list
- alignmentsscope (default : "true") : (true | false) +
- columnwidth (default : "auto") : (auto | number h-unit | namespace | fit) +
- width (default : "auto") : auto | number h-unit
- rowspacing (default : "1.0ex") : (number v-unit) +
- columnspacing (default : "0.8em") : (number h-unit | namespace) +
- rowlines (default : "none") : (none | solid | dashed) +
- columnlines (default : "none") : (none | solid | dashed) +
- columnlines (default : "none") : none | solid | dashed

- frame (default : "none") : none | solid | dashed
- framespacing (default : 0.4em, 0.5ex) : (number h-unit | namedspace) (number v-unit | namedspace)
- equalrows (default : "false") : true | false
- equalcolumns (default : "false") : true | false
- displaystyle (default : "false") : true | false
- side (default : "right") : left | right | leftoverlap | rightoverlap
- minlabelspacing (default : "0.8em") : number h-unit | namedspace

The default setting for “frame”, “rowlines” and columnlines are “none”. For this reason, the rendering of a table with bare “mtable” results in a frame-less invisible table. Only content of the columns are displayed. A 2X3 table is drawn with default values in the example given here.

Example:

Table with default values

```
<m:math display="block"> <m:mtable> <m:mtr> <m:mttd> <m:mi>x</m:mi> </m:mttd>
<m:mttd> <m:mi>y</m:mi> </m:mttd> <m:mttd> <m:mi>z</m:mi> </m:mttd> </m:mtr>
<m:mtr> <m:mttd> <m:mn>1</m:mn> </m:mttd> <m:mttd> <m:mn>2</m:mn> </m:mttd>
<m:mttd> <m:mn>3</m:mn> </m:mttd> </m:mtr> </m:mtable> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

x	y	z
1	2	3

The “mtable” element has large number of attributes and it would be difficult to use all of them. Good news is that we are using only few of them at the most. The dimensional attributes of table frame, rows and columns are largely determined by the implementation of a particular renderer. Also, the default values of key attributes are so chosen that default table architecture is suitable for most of the situations. For example, width controlling attributes “width” and “columnwidth” are set to “auto”. This means that width of the table as a whole and the columns shall be adjusted in accordance with the algorithm of the renderer, whereby they are expected to be set “as needed”. We may ofcourse set specific values whereby content in the table is expected to be wrapped around.

Assignment to table attributes follows a typical format. The reason for special mechanism arises out of the fact that we are required to set some the attributes on individual row or column. The format is simple. It is presented with a “+” sign at the end indicating that the attribute can be applied to individual rows and columns. A general format for columns would expand explained here :

(x y z)+ : It is a sequence of values separated by space. The values can be one or more (i.e. can be repeated) of x,y and z for corresponding columns. Starting from the first, the values are applied till the last column. If there are more values than required, then remaining would be discarded and if there are less than required then the last of the sequence would be repeated.

This means that a large number of attributes such as “rowalign”, “columnalign”, “columnwidth”, “rowspacing”, “columnspacing”, “rowlines” and “columnlines” can be set to individual rows and columns with the valid values as specified above.

The example here illustrates setting of “columnalign” and “rowalign” elements. Note that the impact of “columnalign” is more visible than “rowalign” element.

Example:

Aligning content

```
<m:math display="block"> <m:mtable frame="solid" columnlines="solid"
rowlines="solid" rowalign="top baseline center" columnalign="right center
left"> <m:mtr> <m:mttd> <m:mi>xyx</m:mi> </m:mttd> <m:mttd> <m:mi>y</m:mi>
</m:mttd> <m:mttd> <m:mi>z</m:mi> </m:mttd> </m:mtr> <m:mtr> <m:mttd>
<m:mn>12</m:mn> </m:mttd> <m:mttd> <m:mn>2</m:mn> </m:mttd> <m:mttd>
<m:mn>3111</m:mn> </m:mttd> </m:mtr> </m:mtable> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

xyx	y	z
12	2	3111

"mtr" table row element

Specific attributes : rowalign columnalign denomalign groupalign

Attribute value types

- rowalign (default : inherited) : top | bottom | center | baseline | axis
- columnalign (default : inherited): (left | center | right) +
- groupalign (default : inherited): group-alignment-list-list

Rows are drawn from top to bottom. Entries in the row can be aligned separately on individual row. This row level setting of “rowalign” attribute shall override “rowalign” attribute setting on table level via “mtable” element. The “columnalign” attribute allows specifying individual column alignments for that particular row.

Example:

Alignment setting on individual row

```
<m:math display="block"> <m:mtable frame="solid" columnlines="solid"
rowlines="solid" rowalign="top" columnalign="right"> <m:mtr> <m:mttd>
<m:mi>xyx</m:mi> </m:mttd> <m:mttd> <m:mi>y</m:mi> </m:mttd> <m:mttd>
<m:mi>z</m:mi> </m:mttd> </m:mtr> <m:mtr rowalign="bottom"
columnalign="left"> <m:mttd> <m:mn>12</m:mn> </m:mttd> <m:mttd> <m:mn>2</m:mn>
</m:mttd> <m:mttd> <m:mn>3111</m:mn> </m:mttd> </m:mtr> </m:mtable> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

xyx	y	z
12	2	3111

There appears serious gap in the specification for this element and implementation of the same by the browsers. For this reason, the "mlabeledtr" element is not described. Also related "side" and "minlabelspacing" of "mtable" elements are skipped. Instead, an example with equivalent effect for creating a labeled display is included in the next section titled “Application”.

The implementation of alignment within a cell is also not implemented by browsers at present. As such, "maligngroup" and "malignmark" elements along with associated "groupalign" attribute of "mtable" element are not considered.

Applications

Highlighting rows and columns with dashed line

Control on each column and row is not limited to alignment. The flexibility at the lowest level of creating table is extended to lines bordering table cells as well.

We can highlight a portion of table by assigning a particular row and column with “dashed” lines. This effect can be achieved by specifying individual row and column lines to be “dashed” as against “solid” or “none”. In the table here first row and column lines are set to none, while second row and column lines are set to “dashed”.

Example:

Highlighting rows and columns

```
<m:math display="block"> <m:mtable frame="solid" rowlines="none dashed"
columnlines="solid dashed"> <m:mtr> <m:mttd> <m:mi>x</m:mi> </m:mttd> <m:mttd>
<m:mi>y</m:mi> </m:mttd> <m:mttd> <m:mi>z</m:mi> </m:mttd> </m:mtr> <m:mtr>
<m:mttd> <m:mn>a</m:mn> </m:mttd> <m:mttd> <m:mn>b</m:mn> </m:mttd> <m:mttd>
<m:mn>c</m:mn> </m:mttd> </m:mtr> <m:mtr> <m:mttd> <m:mn>1</m:mn> </m:mttd>
<m:mttd> <m:mn>2</m:mn> </m:mttd> <m:mttd> <m:mn>3</m:mn> </m:mttd> </m:mtr>
</m:mtable> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

x	y	z
a	b	c
1	2	3

Rows of equations

New rows in table can be introduced with “mtr” row element. This behavior is handy in displaying rows of mathematical expressions, using bare table as shown in the example :

Example:

Rows of equations

```
<m:math display="block"> <m:mtable> <m:mtr> <m:mttd> <m:mi>3x + y </m:mi>
</m:mttd> <m:mttd> <m:mo>=</m:mo> </m:mttd> <m:mttd> <m:mn>6</m:mn> </m:mttd>
</m:mtr> <m:mtr> <m:mttd> <m:mn>2x - y </m:mn> </m:mttd> <m:mttd> <m:mo>=
</m:mo> </m:mttd> <m:mttd> <m:mn>4</m:mn> </m:mttd> </m:mtr> </m:mtable>
</m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\begin{array}{rcl} 3x + y & = & 6 \\ 2x - y & = & 4 \end{array}$$

Labelled equation

The label row is convenient to write numbered equation as shown here. This special row is implemented with "mlabelmtr" element.

Example:

Labelled row : equation number

```
<m:math display="block"> <m:mtable> <m:mlabeledtr> <m:mtd> <m:mrow>
<m:mi>E</m:mi> <m:mo>=</m:mo> <m:mrow> <m:mi>m</m:mi> <m:msup>
<m:mi>c</m:mi> <m:mn>2</m:mn> </m:msup> </m:mrow> </m:mtd> <m:mtd>
<m:mtext> (1.1) </m:mtext> </m:mtd> </m:mlabeledtr> </m:mtable> </m:math>
```

Save the file after editing as "test.xml". The display looks like :

mlabeledtr

Matrix

Use of parentheses and table without frame can conveniently be used to represent matrix as illustrated in the example here.

Example:

Matrix

```
<m:math display="block"> <m:mo> ( </m:mo> <m:mtable> <m:mtr> <m:mtd>
<m:mi>x</m:mi> </m:mtd> <m:mtd> <m:mi>y</m:mi> </m:mtd> <m:mtd>
<m:mi>z</m:mi> </m:mtd> </m:mtr> <m:mtr> <m:mtd> <m:mn>1</m:mn> </m:mtd>
<m:mtd> <m:mn>2</m:mn> </m:mtd> <m:mtd> <m:mn>3</m:mn> </m:mtd> </m:mtr>
</m:mtable> <m:mo> ) </m:mo> </m:math>
```

Save the file after editing as "test.xml". The display looks like :

$$\begin{pmatrix} x & y & z \\ 1 & 2 & 3 \end{pmatrix}$$

Writing MathML

We have worked with programs like Dreamweaver and Frontpage to write HTML mark-ups. On similar lines, there are programs to generate MathML mark-ups with built in facilities for error handling and testing. Using specialized programs to generate codes is generally the fastest way to write MathML codes.

Using specialized programs has the obvious advantage, but with certain overheads. It requires that we go through the process of familiarization with the program interface and understand the working philosophy of the same. Though these programs are robust and designed to fit in with all sorts of operating systems and deal with interoperability issues, there remains a remote possibility of mismatch and one has to take appropriate safeguards against difficulties arising out of the same. Worst of this approach is the sense of working within a “block box”, which might spring surprises with regard to what we expect and what the program delivers.

As against above, coding in the native way, using plain text editor has the disadvantages of the requirement of knowing MathML specification and relatively slower pace of coding. The first aspect, though, can be used to advantage as you have to understand the syntax, grammar, underlying features, limitations and possibilities of the MathML, knowing fully well about what can be done and what can not be done with MathML. So, there is less likelihood of a surprise situation.

Further, it is a real possibility that you may actually cut down on the encoding substantially (typing per se) along with a corresponding increase in the coding speed. However, this would require a bit of planning and proficiency in copy (ctrl + c), cut (ctrl + x) and paste (ctrl + v) operations with a block of codes from one place to another.

Features of MathML encoding

Before, we get into the rhythm of encoding, let us discuss the important features of MathML coding and our mathematical content requirement that would help us formulate an efficient approach to write MathML codes.

1: There are 30 elements in total with about 50 specific attributes, apart from common attributes.

2: A MathML coding is sensitive to the context and material being encoded. If your module involves matrix operation, then we might be using the set of table forming elements more frequently. This means that we shall be limited in using elements – may be not more than 10 to 15 of them in a given context (project). Further, we may not require to change the attributes except on few occasions.

3: Mathematical display is a two dimensional representation, but its coding is essentially linear, in which terms are connected with operators from left to right. As a consequence, codes are written progressively from left to right of the mathematical expressions and equations.

4: The vertical expansion of mathematical expression is controlled by a central element. For example a vertical stack may be structured on a form, implemented by “mfrac” element.

Illustration

To illustrate different aspects of MathML encoding and logical sequence in which code lines are written, we undertake to write MathML codes for an equation given here :

$$x = \left[32000 \left(1 - \frac{5}{100} \right) \left(1 + \frac{10}{100} \right) \left(1 + \frac{\frac{25}{2}}{100} \right) \right]$$

Step 1 : Observe the content of the equation from left to right. Identify terms requiring expansion in vertical directions. In the given equation, there are three such terms. It is evident that “mfrac” element suits the requirement in hand. The last of three “mfrac” based expression, matter of fact, would require nested use of “mfrac” element.

Step 2 : Start from the left and write or insert basic construct of “mi” element (line 2).

```
1: <m:math display="block"> 2: <m:mi> x </m:mi> 3:
</m:math>
```

Step 3 : Copy “mi” element construct (line 2) and paste on the following line (line 3). Change “i” to “o” in order to change the initial and final tags to “mo” tags and change “x” by “=” so that code looks like :

```
1: <m:math display="block"> 2: <m:mi> x </m:mi> 3:
<m:mo> = </m:mo> </m:math>
```

Step 4 : In this step, using the technique of Step 2, add following 5 lines at the end (4 to 8):

```
1: <m:math display="block"> 2: <m:mi> x </m:mi> 3:
<m:mo> = </m:mo> 4: <m:mi> x </m:mi> 5: <m:mi> x
</m:mi> 6: <m:mi> x </m:mi> 7: <m:mi> x </m:mi> 8:
<m:mi> x </m:mi> </m:math>
```

Use appropriate token elements with identifiers, operators and numbers by modifying the newly added lines :

```
1: <m:math display="block"> 2: <m:mi> x </m:mi> 3:
<m:mo> = </m:mo> 4: <m:mo> [ </m:mo> 5: <m:mn>
3200 </m:mn> 6: <m:mo> ( </m:mo> 7: <m:mn> 1
</m:mn> 8: <m:mo> - </m:mo> </m:math>
```

What you have coded till now : $x = [3200(1 -$

What is to be coded ultimately :

$$x = \left[32000 \left(1 - \frac{5}{100} \right) \left(1 + \frac{10}{100} \right) \left(1 + \frac{\frac{25}{2}}{100} \right) \right]$$

Step 4 : In this step, we shall handle the vertical term with “mfrac” element. Insert the basic block of “mfrac” element or type at the end (9 to 12) as shown :

```

1: <m:math display="block"> 2: <m:mi> x </m:mi> 3:
<m:mo> = </m:mo> 4: <m:mo> [ </m:mo> 5: <m:mn>
3200 </m:mn> 6: <m:mo> ( </m:mo> 7: <m:mi> 1
</m:mi> 8: <m:mo> - </m:mo> 9: <m:mfrac> 10:
<m:mn> a </m:mn> 11: <m:mn> b </m:mn> 12:
</m:mfrac> </m:math>

```

Step 5 : Change the value of “mfrac” element’s argument as given in the equation. Copy line 6 twice and modifying the same to put a closing and opening parentheses (13 and 14). Further, add lines 15 and 16 by copying and modifying lines 7 and 8 :

```

1: <m:math display="block"> 2: <m:mi> x </m:mi> 3:
<m:mo> = </m:mo> 4: <m:mo> [ </m:mo> 5: <m:mn>
3200 </m:mn> 6: <m:mo> ( </m:mo> 7: <m:mn> 1
</m:mn> 8: <m:mo> - </m:mo> 9: <m:mfrac> 10:
<m:mn> 5 </m:mn> 11: <m:mn> 100 </m:mn> 12:
</m:mfrac> 13: <m:mo> ) </m:mo> 14: <m:mo> (
</m:mo> 15: <m:mn> 1 </m:mn> 16: <m:mo> + </m:mo>
</m:math>

```

Step 5 : Looking at the equation, it is easy to note that we need to have two fraction term with nominator and denominator. Thus, copy the block to display vertical term i.e. copy and paste “mfrac” code lines 9 to 12 and modify according to the requirement (new lines 17 to 20). Copy and paste code lines 13 to 16 and modify as required (new lines 21 to 24). Finally copy and paste code lines 9 to 12 again as shown (new lines 25 to 28).

```

1: <m:math display="block"> 2: <m:mi> x </m:mi> 3:
<m:mo> = </m:mo> 4: <m:mo> [ </m:mo> 5: <m:mn>
3200 </m:mn> 6: <m:mo> ( </m:mo> 7: <m:mn> 1
</m:mn> 8: <m:mo> - </m:mo> 9: <m:mfrac> 10:
<m:mn> 5 </m:mn> 11: <m:mn> 100 </m:mn> 12:
</m:mfrac> 13: <m:mo> ) </m:mo> 14: <m:mo> (
</m:mo> 15: <m:mn> 1 </m:mn> 16: <m:mo> + </m:mo>
17: 9: <m:mfrac> 18: 10: <m:mn> 10 </m:mn> 19: 11:
<m:mn> 100 </m:mn> 20: 12: </m:mfrac> 21: 13:

```

```
<m:mo> ) </m:mo> 22: 14: <m:mo> ( </m:mo> 23: 15:
<m:mn> 1 </m:mn> 24: 16: <m:mo> + </m:mo> 25: 9:
<m:mfrac> 26: 10: <m:mn> 5 </m:mn> 27: 11: <m:mn>
100 </m:mn> 28: 12: </m:mfrac> </m:math>
```

What you have coded till now : $x = \left[3200 \left(1 - \frac{5}{100} \right) \left(1 + \frac{10}{100} \right) \left(1 + \frac{5}{100} \right) \right]$

What is to be coded ultimately :

$$x = \left[32000 \left(1 - \frac{5}{100} \right) \left(1 + \frac{10}{100} \right) \left(1 + \frac{\frac{25}{2}}{100} \right) \right]$$

Step 6 : Note that last “mfrac” display requires another “mfrac” implementation. Its numerator itself is a “mfrac” display. Thus, we would require to change the line 26 placed near bottom with a block of “mfrac” codes that shall display “25/2”. For this we replace code line 26 with code block consisting of code lines 9 to 12. In order to make this block as the numerator of the parent “mfrac” element, we need to enclose this block with “mrow” element so that “mfrac” element considers the block as one argument. Change the “mo’ content (lines 10 and 11 at the bottom) to reflect the ratio as 25/2. Finally, add a closing parenthesis and a bracket at the end (lines 29 and 30) as shown.

```
13: <m:mo> ) </m:mo> 14: <m:mo> ( </m:mo> 15:
<m:mn> 1 </m:mn> 16: <m:mo> + </m:mo> 17: 9:
<m:mfrac> 18: 10: <m:mn> 10 </m:mn> 19: 11: <m:mn>
100 </m:mn> 20: 12: </m:mfrac> 21: 13: <m:mo> )
</m:mo> 22: 14: <m:mo> ( </m:mo> 23: 15: <m:mn> 1
</m:mn> 24: 16: <m:mo> + </m:mo> 25: 9: <m:mfrac>
---- <m:mrow> 9: <m:mfrac> 10: <m:mn> 25 </m:mn>
11: <m:mn> 2 </m:mn> 12: </m:mfrac> </m:mrow> ----
- 27: 11: <m:mn> 100 </m:mn> 28: 12: </m:mfrac>
29: <m:mo> ) </m:mo> 30: <m:mo> ] </m:mo>
</m:math>
```

Step 7 : Save the file as test.xml. The code (after renumbering for reference purpose) at this stage looks like :

```

13: <m:mo> ) </m:mo> 14: <m:mo> ( </m:mo> 15:
<m:mn> 1 </m:mn> 16: <m:mo> + </m:mo> 17:
<m:mfrac> 18: <m:mn> 10 </m:mn> 19: <m:mn> 100
</m:mn> 20: </m:mfrac> 21: <m:mo> ) </m:mo> 22:
<m:mo> ( </m:mo> 23: <m:mn> 1 </m:mn> 24: <m:mo> +
</m:mo> 25: <m:mfrac> 26: <m:mrow> 27: <m:mfrac>
28: <m:mn> 25 </m:mn> 29: <m:mn> 2 </m:mn> 30:
</m:mfrac> 31: </m:mrow> 32: <m:mn> 100 </m:mn>
33: </m:mfrac> 34: <m:mo> ) </m:mo> 35: <m:mo> ]
</m:mo> </m:math>

```

What you have coded till now :

$$x = \left[3200 \left(1 - \frac{5}{100} \right) \left(1 + \frac{10}{100} \right) \left(1 + \frac{5}{100} \right) \right]$$

What is to be coded ultimately :

$$x = \left[32000 \left(1 - \frac{5}{100} \right) \left(1 + \frac{10}{100} \right) \left(1 + \frac{\frac{25}{2}}{100} \right) \right]$$

Step 8 : Comparing what we have achieved so far and what is the expected, we need few tweaking here and there. First we see that the brackets have not grown to the vertical height of the terms, composing the equation. In order to do this, we club the part of equation in the brackets within “mrow” tags. Thus, we insert “mrow” tag before line 4 and at the end of the code after line 35. Also, the bars of the three central "mfrac" elements have to be distinguished from the nested one. We use "linethickness" attribute on "mfrac" element and set the same to "medium". The final code and display are shown in the example.

Example:

Indices element : mroot

```

<m:math display="block"> <m:mi> x </m:mi> <m:mo> =
</m:mo> <m:mrow> <m:mo> [ </m:mo> <m:mn> 32000
</m:mn> <m:mo> ( </m:mo> <m:mn> 1 </m:mn> <m:mo> -
</m:mo> <m:mfrac linethickness="2"> <m:mn> 5
</m:mn> <m:mn> 100 </m:mn> </m:mfrac> <m:mo> )

```

```

</m:mo> <m:mo> ( </m:mo> <m:mn> 1 </m:mn> <m:mo> +
</m:mo> <m:mfrac linethickness="2"> <m:mn> 10
</m:mn> <m:mn> 100 </m:mn> </m:mfrac> <m:mo> )
</m:mo> <m:mo> ( </m:mo> <m:mn> 1 </m:mn> <m:mo> +
</m:mo> <m:mfrac linethickness="2"> <m:mrow>
<m:mfrac> <m:mn> 25 </m:mn> <m:mn> 2 </m:mn>
</m:mfrac> </m:mrow> <m:mn> 100 </m:mn> </m:mfrac>
<m:mo> ) </m:mo> <m:mo> ] </m:mo> </m:mrow>
</m:math>

```

Save the file after editing as “test.xml”. The display looks like :

$$x = \left[32000 \left(1 - \frac{5}{100} \right) \left(1 + \frac{10}{100} \right) \left(1 + \frac{\frac{25}{2}}{100} \right) \right]$$

Speeding up with MathML

Surprisingly, we are not always aware of things available to us. MS Word, for example, has the capability to reproduce a chunk of text with a stroke of key combination. This appears a great facility, particularly for speeding up writing any mark up codes, including MathML. This facility, though available all the time, is hardly ever used in normal text editing session with MS Word.

Using the facility to create macros in MS Word is simple. Matter of fact, this is one of the reasons for recommending an advanced text editing software like MS Word as against bare editors like Notepad for writing plane text MathML code lines.

We need to create macros for writing a set of code lines for the basic coding forms for most of the frequently used element and assign the same to a keyboard combination like ALT a, ALT b etc. Subsequently, when we need to write the code lines for MathML element in the basic form, we shall require to simply use the key combination and the code lines will be reproduced. It is warned that we must provide sufficient space between existing code or text lines so that the reproduced text by macros do not mix up. This techniques, involving macros, has the advantage that once a correct format is recorded, it is less likely that there would be any syntax error.

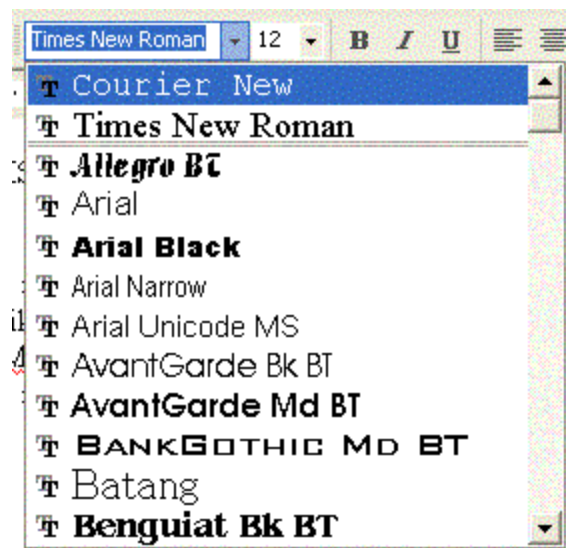
Creating macros for MathML basic forms of elements

In this section, the exact steps to create a macro with few code lines involving “mfrac” element are given. Before we proceed, we must realize that choosing a particular size and font to write the codes in macros is extremely important. The renderer accepts MathML in mono font for the code lines. What it means that we should encode our markups in one font only. A mix of many fonts results from “copy” and “paste” operations from different sources. We must avoid font mixing so that the renderer does not report error, while displaying the content.

Now, follow the steps to create a macro for the basic form of “mfrac” element :

Step 1: Open MS Word. Create a new document. Save the document as “test.doc”, ensuring that “save as type” is a word document. Note that we can **not** record and store macros in plain format, which is used for encoding MathML.

Step 2: Select “Courier New” from the font drop down as shown here :
Font selection

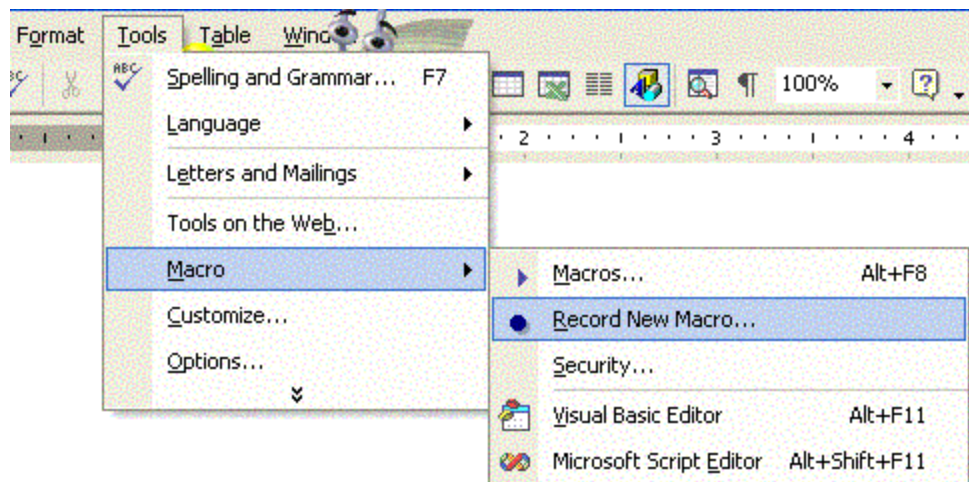


Select font size 12 from the adjacent drop down.

Step 3 : Write code lines for the basic form of the “mfrac” element as shown here :

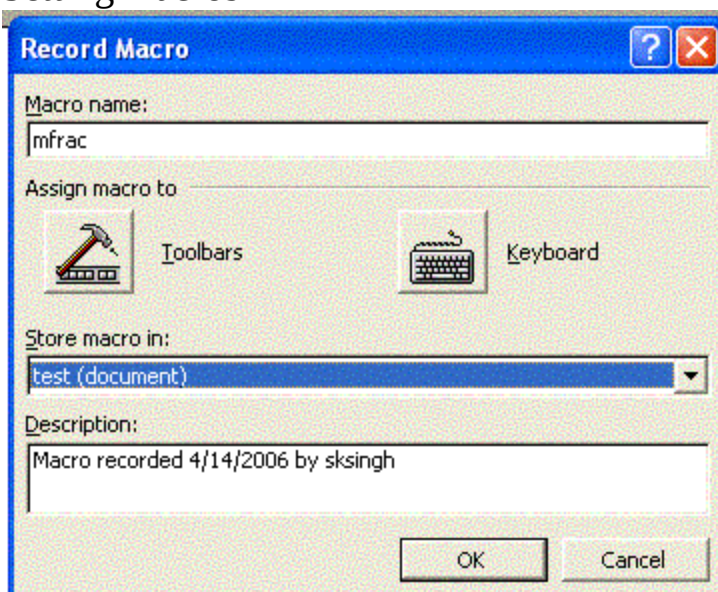
```
<m:mfrac> <m:mi> x </m:mi> <m:mi> y </m:mi>  
</m:mfrac>
```

Step 4 : Select “Tools” from the top tool bar of MS Word and choose “Record New Macro” as shown below :
Recording macros



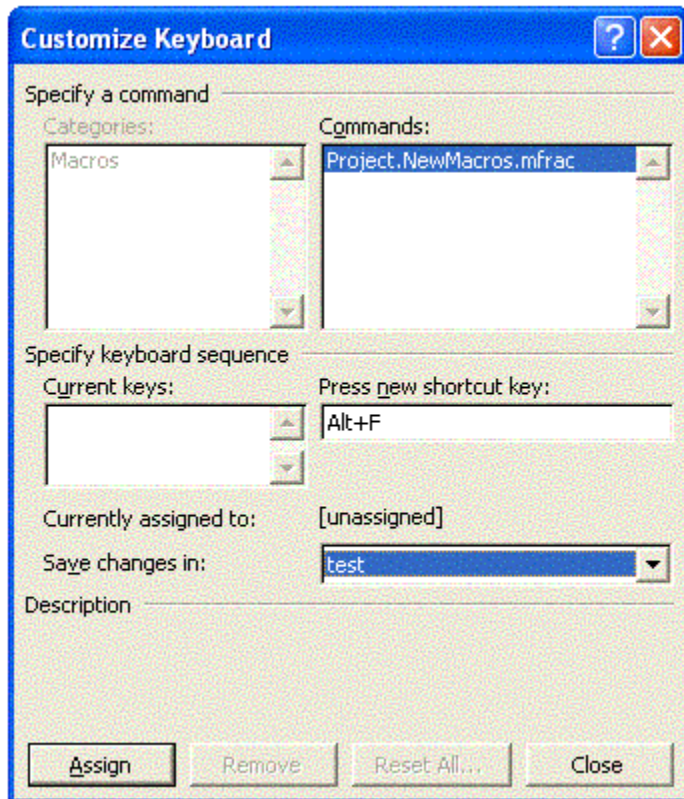
Step 5 : “Record macro” dialog box appears. In the “Macro name” give the name of macro, say “mfrac”. In the “Store macro in” select the name of document “test document” or the name of the document in which you are recording macro.

Setting macros



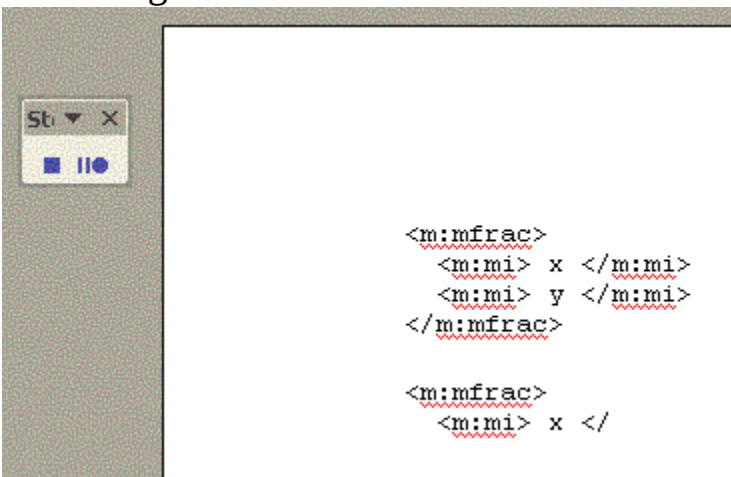
Step 6 : Press “keyboard” button. The “Customize Keyboard” dialog box appears. Put the cursor in the “Press new shortcut key” box (if not placed automatically). Press “Alt and f” to record the desired key combination to run the macro for “mfrac” code lines. Note that “Alt + F” text with capital “F” is automatically placed inside the box. In the “unassigned” box, select “test”.

Assigning key combination



Step 7 : Press “Assign” button and then “Close” button. Macro recording has started. A pair of buttons are displayed on the margin of the document as shown in the figure below. One of the button is “Stop” button, while the other is “Pause” button for macro recording.

Recording code lines



Now, looking at the code lines written in step 3, type exactly the same after one or two lines. Do not bother for key and cursor movements. It is advisable that we maintain the required indenting say between “mfrac” tag and “mi” tag. When the code lines have been reproduced, press “Stop” button to stop the recording.

Step 8 : You are now ready to use the macro. Hold “Alt” key and simultaneously press “F” key. The code lines as recorded are reproduced instantly as shown here :

```
<m:mfrac> <m:mi> x </m:mi> <m:mi> y </m:mi>  
</m:mfrac>
```

Macro recording is a powerful tool to reproduce a set of code lines. However, this facility is available in the normal document mode and not in the plane text mode. As such, codes are required to be first written in a document file, say “test.doc”. Once coding is complete in a major way, we need to save the file in plane text with Unicode character format (UTF -8) for renderer to display the content. It is implied here that a repository of macros corresponding to the basic forms of element encoding be prepared in advance in order to use the same, when required.

In the following section, we shall illustrate the use of macros and few improvisations to encode “Cauchy-Schwarz” inequality.

“Cauchy-Schwarz” inequality

Understanding of the underlying presentation framework of MathML is just one aspect of learning process. Equally important aspect is to develop the ability to convert presentation requirement in terms of codes. There is no universal set of algorithms that would help to successfully translate visual presentation requirement into MathML code lines in a mechanical manner - unless aided by specialized program or software. Unaided, this may be achieved by developing ability to visualize the presentation requirements in some logical modular work pieces like :

1. Break the presentation requirement in smaller components

2. Write building blocks of code with the help of macros
3. Copy and paste identical components and modify
4. Combine the code lines to make a “whole”

“Cauchy-Schwarz” inequality is shown here :

$$\left(\sum_{k=1}^n a_k b_k \right)^2 \leq \left(\sum_{k=1}^n a_k^2 \right) \left(\sum_{k=1}^n b_k^2 \right)$$

A visual inspection of the formula suggests that this inequality is built of three large blocks of similar composition. The first block is raised to a power of “2”. This can be implemented with “msup” element. Thus, we start with using macro for “msup” element :

```
<m:msup> <m:mi> x </m:mi> <m:mi> y </m:mi>
</m:msup>
```

The superscript is changed to “2” as required. A code line for element “mo” is written using macro for “mo” element and its content is changed to parenthesis :

```
<m:math display="block"> <m:mo> ( </m:mo> <m:msup>
<m:mi> x </m:mi> <m:mn> 2 </m:mn> </m:msup>
</m:math>
```

The base of the superscript is the large block, involving summation sign. The base is built upon summation sign with “over” and “under” limits implemented by “munderover” element. We write “munderover” basic code lines, using macro to replace the “base” of superscript i.e. “x” in the basic format reproduced by macro in the above step.

```
<m:math display="block"> <m:mo> ( </m:mo> <m:msup>
<m:munderover> <m:mo> &int; </m:mo> <m:mi>x</m:mi>
<m:mi>y</m:mi> </m:munderover> <m:mn> 2 </m:mn>
</m:msup> </m:math>
```

We replace integer symbol with summation symbol “ $\sum$ ” and change the limits as required. Note that we need to bunch the underscript with “mrow” tags to work as one of the argument of the “msup” element.

```
<m:math display="block"> <m:mo> ( </m:mo> <m:msup>
<m:munderover> <m:mo>&sum;</m:mo> <m:mrow> <m:mi>
k </m:mi> <m:mo> = </m:mo> <m:mn> 1</m:mn>
</m:mrow> <m:mn>n</m:mn> </m:munderover> <m:mn> 2
</m:mn> </m:msup> </m:math>
```

Now, we add codes for a_k and b_k , which can be implemented with “msub” element. We produce two sets of “msub” codes with macro and place them just after </m:munderover>. Finally, we also insert a “mo” line to encode for closing parenthesis. Putting these code lines ahead of superscript element line <m:mi> 2 </m:mi> is important as everything in totality is raised to the power “2”. For this reason, everything other than superscript element line is required to be bunched together with a pair of “mrow” tags as shown here :

```
<m:math display="block"> <m:mo> ( </m:mo> <m:msup>
<m:mrow> <m:munderover> <m:mo>&sum;</m:mo>
<m:mrow> <m:mi> k </m:mi> <m:mo> = </m:mo> <m:mn>
1</m:mn> </m:mrow> <m:mn>n</m:mn> </m:munderover>
<m:msub> <m:mi> a </m:mi> <m:mi> k </m:mi>
</m:msub> <m:msub> <m:mi> b </m:mi> <m:mi> k
</m:mi> </m:msub> <m:mo> ) </m:mo> </m:mrow>
<m:mn> 2 </m:mn> </m:msup> </m:math>
```

This completes the coding for the first left hand side block of the “Cauchy-Schwarz” inequality. At this stage, the display in the browser looks like :

$$\left(\sum_{k=1}^n a_k b_k \right)^2$$

Now coding for two other large block is easy. We need to replicate the above code block twice and do the modification as required. Finally three

blocks are joined together with the help of an “inequality” operator “≤”. The display, now, looks like :

$$\left(\sum_{k=1}^n a_k b_k\right)^2 \leq \left(\sum_{k=1}^n a_k^2\right) \left(\sum_{k=1}^n b_k^2\right)$$

We observe that the parenthesis of the first term has not grown to the expected height to cover the highest term. Moreover, the parentheses are unequal in size. To achieve the equality in size and proper coverage, we enclose all expressions involving parentheses within a pair of “mrow” tags and a “mstyle” pairs of tags to display “true”. The complete code and display is illustrated in the example below :

Example:

“Cauchy-Schwarz” inequality

```
<m:math display="block"> <m:mstyle
displaystyle='true'> <m:mrow> <m:msup> <m:mrow>
<m:mo> ( </m:mo> <m:munderover> <m:mo>&sum;</m:mo>
<m:mrow> <m:mi> k </m:mi> <m:mo> = </m:mo> <m:mn>
1</m:mn> </m:mrow> <m:mn>n</m:mn> </m:munderover>
<m:msub> <m:mi> a </m:mi> <m:mi> k </m:mi>
</m:msub> <m:msub> <m:mi> b </m:mi> <m:mi> k
</m:mi> </m:msub> <m:mo> ) </m:mo> </m:mrow>
<m:mn> 2 </m:mn> </m:msup> <m:mo> &le; </m:mo>
<m:mrow> <m:mo> ( </m:mo> <m:munderover>
<m:mo>&sum;</m:mo> <m:mrow> <m:mi> k </m:mi>
<m:mo> = </m:mo> <m:mn> 1</m:mn> </m:mrow>
<m:mn>n</m:mn> </m:munderover> <m:msup> <m:mrow>
<m:msub> <m:mi> a </m:mi> <m:mi> k </m:mi>
</m:msub> </m:mrow> <m:mn> 2 </m:mn> </m:msup>
<m:mo> ) </m:mo> </m:mrow> <m:mrow> <m:mo> (
</m:mo> <m:munderover> <m:mo>&sum;</m:mo> <m:mrow>
<m:mi> k </m:mi> <m:mo> = </m:mo> <m:mn> 1</m:mn>
</m:mrow> <m:mn>n</m:mn> </m:munderover> <m:msup>
```

```
<m:mrow> <m:msub> <m:mi> b </m:mi> <m:mi> k  
</m:mi> </m:msub> </m:mrow> <m:mn> 2 </m:mn>  
</m:msup> <m:mo> ) </m:mo> </m:mrow> </m:mrow>  
</m:mstyle> </m:math>
```

Save the file after editing as “test.xml”. The display looks like :

$$\left(\sum_{k=1}^n a_k b_k \right)^2 \leq \left(\sum_{k=1}^n a_k^2 \right) \left(\sum_{k=1}^n b_k^2 \right)$$

Unicode character set (A-H)

The Unicode character set is arranged in the alphabetical order of corresponding entity reference names. We may use the characters like “ β ” directly (by copying and pasting the character) or may opt to reference them with entity reference as “`β`” within the presentation token elements.

A					
aacute	á	Aacute	Á	ac	→
acirc	â	acute	´	acy	α
af		afr	𝐚	Afr	ℳ
alpha	α	amacr	ā	Amacr	Ā
and	∧	andand	⋈	alpha	α
ang	∠	ange	⋈	angle	∠
angmsdac	⚡	angmsdad	⚡	angmsdae	⚡
angrtvbd	⚡	angrtvbd	⚡	angsph	⋈
Aogon	Ą	aopf	Ⓐ	Aopf	Ⓐ
apE	≈	apid	≈	apos	’
aring	å	Aring	Å	ascr	“
asyp	⋈	Atilde	Ã	atilde	ã
awconint	ℳ	awint	ℳ		
B					
backcong	≡	backepsilon	ε	backprime	′
Barv	⚡	barvee	⚡	barwed	⋈
bcong	≡	bcy	Ბ	Bcy	Ბ
bemptyv	∅	bepsi	ε	bernou	ℬ
between	ℳ	Bfr	𝐁	bfr	𝐁
bigodot	⊙	bigoplus	⊕	bigotimes	⊗
bigtriangledown	▽	bigtriangleup	△	biguplus	⊕
blacklozenge	◆	blacksquare	■	blacktriangle	▲
blank	␣	blk12	␣	blk14	␣
bnequiv	≡	bnot	¬	bNot	≡
bottom	⊥	bowtie	⋈	boxbox	⊠
Breve	˘	breve	˘	brvbar	˘
bsim	∼	bsime	⋈	bsol	∖
bullet	•	bump	⋈	bumpE	⋈
C					
cacute	´	Cacute	Ć	cap	∩
capcap	⋈	capcup	⋈	capdot	⋈
caron	˘	Cayleys	ℭ	ccaps	⋈
Ccirc	Ĉ	ccirc	ĉ	Cconint	⋈
Cdot	Ċ	cedil	¸	Cedilla	¸
CenterDot	·	cfr	ℭ	Cfr	ℭ
checkmark	✓	chi	χ	cir	○
circlearrowright	↻	circledast	⊗	circledcirc	⊙
circledS	Ⓢ	CircleMinus	⊖	CirclePlus	⊕
cirfnint	ℳ	cirmid	ℳ	cirscir	⊙

clubs	♣	clubsuit	♣	Colon	::
coloneq	⋈	comma	,	commat	@
complexes	ℂ	cong	≅	congdots	≐
ContourIntegral	∮	Copf	ℭ	copf	ℭ
copysr	Ⓢ	CounterClockwiseContourIntegral	∮	Cross	×
csup	⊃	csup	⊃	csup	⊃
cudarr	↴	cuepr	↵	cuesc	↵
cup	∪	Cup	⊃	cupbrcap	⊃
cupdot	⊃	cupor	⊃	cups	∪
curlyeqsucc	↷	curlyvee	⋈	curlywedge	⋈
cuvee	⋈	cuwed	⋈	cwconint	⋈
D					
Dagger	†	daleth	⧫	dagger	†
dashv	⊣	Dashv	⊣	dbkarow	↔
Dcy	Д	dcy	∂	dd	∂
deg	°	Del	∇	Delta	Δ
diam	◇	diamond	◇	Diamond	◇
digamma	ƒ	disin	∉	div	÷
divonx	*	djcy	ђ	DJcy	Ђ
dollar	\$	Dopf	℔	dopf	℔
doteq	⋈	doteqdot	⋈	DotEqual	⋈
doublebarwedge	⋈	DoubleContourIntegral	⋈	DoubleDot	⋈
DoubleLeftTee	⊣	DoubleLongLeftArrow		DoubleLongLeftRightArrow	↔
DoubleUpArrow	⬆	DoubleUpDownArrow	↕	DoubleVerticalBar	
downdownarrows	⇓	downharpoonleft	↵	downharpoonright	↵
DownTeeArrow	⬆	drbkarow	↔	drcorn	┐
DScy	Ѕ	dscy	ѕ	dsol	/
dtri	▽	dtrif	▼	duarr	↕
E					
eacute	é	easter	⚡	ecaron	ě
Ecirc	Ê	ecirc	ê	ecolon	⋈
Edot	Ẹ	edot	ė	eDot	⋈
efr	ℰ	eg	≡	egrave	è
el	ℓ	Element	∈	ell	ℓ
emacr	ē	empty	∅	emptyset	∅
emsp		emsp13		emsp14	
eogon	ė	Eogon	Ė	eopf	ℭ
eplus	⋈	epsi	ε	epsiv	ε
eqslantgtr	≷	eqslantless	≶	Equal	==
Equilibrium	⇌	equiv	≡	equivDD	≡
escr	ℰ	Escr	ℰ	esdot	⋈
eth	ð	ETH	Ð	euml	ë
Exists	∃	expectation	ℰ	ExponentialE	ℰ
F					

fallingdotseq	$\fallingdotseq$	Fcy	Φ	fcy	ϕ
ffllig	$\ffl$	ffr	$\mathfrak{f}$	Ffr	$\mathfrak{F}$
flig	$\fl$	fnof	f	Fopf	$\mathbb{F}$
fork	$\pitchfork$	forkv	$\mathcal{M}$	Fouriertrf	$\mathcal{F}$
frac56	$\frac{5}{6}$	frac58	$\frac{5}{8}$	frac78	$\frac{7}{8}$
G					
gacute	$\acute{g}$	Gamma	Γ	gamma	γ
gbreve	$\breve{g}$	Gbreve	$\mathring{G}$	Gcedil	$\mathring{G}$
Gcy	Γ	Gdot	$\dot{G}$	gdot	$\dot{g}$
gEl	$\gtrsim$	geq	$\geq$	geqq	$\geq\geq$
Gg	$\gg$	gg	$\gg$	ggg	$\ggg$
gl	$\gtrless$	gla	$\gtrless$	glE	$\gtrless$
gneqq	$\ngtr$	gnsim	$\napprox$	grave	$\grave{\cdot}$
GreaterFullEqual	$\geq$	GreaterLess	$\gtrless$	GreaterTilde	$\gtrsim$
Gt	$\gg$	gtcc	$\triangleright$	gtcir	$\triangleright\circ$
gtdot	$\triangleright\cdot$	gtlPar	$\nparallel$	gtquest	$\triangleright?$
gtreqless	$\gtrless$	gtreqqless	$\gtrless$	gtrless	$\gtrless$
H					
hairsp		Hacek	$\ˇ$	half	$\frac{1}{2}$
harr	$\leftrightarrow$	hArr	$\Leftrightarrow$	harrcir	$\leftrightarrow\circ$
Hcirc	$\mathring{H}$	hcirc	$\mathring{h}$	heartsuit	$\heartsuit$
HilbertSpace	$\mathcal{H}$	homtht	$\doteq$	hookleftarrow	$\hookleftarrow$
HorizontalLine	---	Hscr	$\mathcal{H}$	hslash	$\hbar$

Unicode character set (I-Q)

The Unicode character set is arranged in the alphabetical order of corresponding entity reference names. We may use the characters like “ β ” directly (by copying and pasting the character) or may opt to reference them with entity reference as “[β](#)” within the presentation token elements.

I						
Iacute	Í	ic		icirc	î	Icin
icy	ı	Idot	İ	iecy	ë	IEc
Ifr	ℑ	Igrave	Ì	igrave	ì	iiin
ijlig	Ĳ	Im	ℑ	Imacr	Ī	ima
imagpart	ℑ	imath	ℓ	imof	∞	Impl
infin	∞	inodot	ℓ	int	∫	In
Integral	∫	intercal	⋮	Intersection	∩	intla
iocy	ë	Iogon	Ĭ	iogon	ï	iot
isin	∈	isindot	€	isinE	€	isir
Itilde	Ĩ	itilde	ĩ	iukcy	і	Iuku
J						
jcirc	Ĵ	jcy	Ї	jsercy	Ј	Jser
K						
kappa	κ	kappav	⋈	Kcedil	Ƙ	kced
kgreen	κ	khcy	х	KHcy	Х	KJcy
L						
lacute	ĺ	Lacute	Ĺ	laemptyv	∅	lagr
lambda	λ	Lang	⟨	lang	⟨	lang
Laplacetrif	ℒ	laquo	«	larr	←	Larr
larrb	↵	larrbfs	↔	larrfs	↔	larrf
larrtl	↵	lbrk	[	lbrace	{	lbra
lcaron	Ľ	Lcaron	Ļ	Lcedil	Ł	lced
Lcy	Л	lcy	л	ldquo	“	ldqu
LeftAngleBracket	⟨	Leftarrow	⇐	leftarrow	←	LeftA
leftarrowtail	↵	LeftCeiling	⌈	LeftDoubleBracket	⌈	LeftDownT
LeftFloor	⌋	leftharpoondown	↘	leftharpoonup	↙	leftlefta
LeftFloor	⌋	leftharpoondown	↘	leftharpoonup	↙	leftlefta
leftrightharpoons	⇔	leftrightsquigarrow	↻	LeftRightVector	↔	LeftT
leftthreetimes	≍	LeftTriangle	◁	LeftTriangleBar	◁	LeftTrian
leg	≦	lEg	≦	leq	≤	leq
lesseqgtr	≧	lesseqqgtr	≧	LessEqualGreater	≧	LessFul
lessgtr	≧	lesssim	≧	LessTilde	≧	lflo
lharu	↵	ljcy	Љ	LJcy	Љ	LJ
llcorner	⌞	Lleftarrow	⇐	llhard	⇐	Lmic
lne	≧	lnE	≧	lneq	≧	lneq
looparrowleft	↶	looparrowright	↷	lopar	⌈	lowt
loz	◊	lpar	(	lrarr	↔	lrcor
lscr	ℓ	Lscr	ℒ	lsh	↵	Lsh
lsquo	‘	lsauor	.	lstrok	ł	Lstr

$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
ltdot	$\cdot$	lthree	$\cdot$	ltimes	$\cdot$	ltr
M						
macr	$\cdot$	male	$\cdot$	malt	$\cdot$	malt
mapstodown	$\cdot$	mapstoleft	$\cdot$	mapstoup	$\cdot$	marl
mcy	$\cdot$	mdash	$\cdot$	mDDot	$\cdot$	measure
Mfr	$\cdot$	mfr	$\cdot$	mho	$\cdot$	mic
midcir	$\cdot$	middot	$\cdot$	minus	$\cdot$	mini
MinusPlus	$\cdot$	mldr	$\cdot$	mnplus	$\cdot$	mod
mstpos	$\cdot$	mu	$\cdot$	multimap	$\cdot$	mun
N						
nabla	$\cdot$	macute	$\cdot$	Nacute	$\cdot$	nar
napid	$\cdot$	napos	$\cdot$	napprox	$\cdot$	nat
nbsp		nbump	$\cdot$	nbumpe	$\cdot$	nca
Ncedil	$\cdot$	ncedil	$\cdot$	ncong	$\cdot$	ncong
Ncy	$\cdot$	ndash	$\cdot$	ne	$\cdot$	near
nearrow	$\cdot$	nedot	$\cdot$	nequiv	$\cdot$	nese
NestedLessLess	$\cdot$	NewLine		nexist	$\cdot$	nexi
ngeqslant	$\cdot$	nges	$\cdot$	ngsim	$\cdot$	ng
nharr	$\cdot$	nhArr	$\cdot$	ni	$\cdot$	niv
nlArr	$\cdot$	nlarr	$\cdot$	nldr	$\cdot$	nll
nLeftrightarrow	$\cdot$	nletrightarrow	$\cdot$	nleq	$\cdot$	nlec
nless	$\cdot$	nlsim	$\cdot$	nlt	$\cdot$	nlt.
NoBreak		NonBreakingSpace		Nopf	$\cdot$	no
NotDoubleVerticalBar	$\cdot$	NotElement	$\cdot$	NotEqual	$\cdot$	NotEe
NotGreaterLess	$\cdot$	NotGreaterSlantEqual	$\cdot$	NotGreaterTilde	$\cdot$	not
NotLess	$\cdot$	NotLessGreater	$\cdot$	NotLessSlantEqual	$\cdot$	NotLess
NotPrecedes	$\cdot$	NotPrecedesSlantEqual	$\cdot$	NotReverseElement	$\cdot$	NotRight'
NotSquareSupersetEqual	$\cdot$	NotSubset	$\cdot$	NotReverseElement	$\cdot$	NotSubse
NotSuperset	$\cdot$	NotSupersetEqual	$\cdot$	NotTilde	$\cdot$	NotTild
npar	$\cdot$	nparallel	$\cdot$	npr	$\cdot$	npre
nrArr	$\cdot$	nrightarrow	$\cdot$	nRightarrow	$\cdot$	nrt
nscue	$\cdot$	nsim	$\cdot$	nsime	$\cdot$	nsim
nsub	$\cdot$	nsubE	$\cdot$	nsube	$\cdot$	nsub
nsucc	$\cdot$	nsup	$\cdot$	nsupE	$\cdot$	nsu
nsupseteqq	$\cdot$	ntgl	$\cdot$	Ntilde	$\cdot$	ntil
ntrianglelefteq	$\cdot$	ntriangleright	$\cdot$	ntrianglerighteq	$\cdot$	nu
numsp		nVDash	$\cdot$	nVdash	$\cdot$	nvD:
nvgt	$\cdot$	nvHarr	$\cdot$	nvlArr	$\cdot$	nv
nwarr	$\cdot$	nwArr	$\cdot$	nwarrow	$\cdot$	nv
O						
Oacute	$\cdot$	oacute	$\cdot$	oast	$\cdot$	oci
Ocy	$\cdot$	ocy	$\cdot$	odash	$\cdot$	odbl
OElig	$\cdot$	oelig	$\cdot$	ogon	$\cdot$	ogra

oint	§	ollar	⊙	omacr	ō	Oma
ominus	⊖	OpenCurlyDoubleQuote	“	OpenCurlyQuote	‘	oph
order	ο	orderof	ο	ordf	ª	ord
oscr	ο	Oslash	Ø	oslash	ø	osc
otimes	⊗	Ouml	Ö	ouml	ö	ovb
OverBar	¯	OverBrace	⏟	OverParenthesis	⏟	
P						
par		para	¶	parallel		par
Pcy	Π	percnt	%	period	.	perr
pertenk	‰	Phi	Φ	phi	φ	phi
Pi	Π	pi	π	pitchfork	⌵	pi
piv	ϐ	planckh	ℏ	plankv	ℏ	plu
plusmn	±	pm	±	Poincareplane	ℋ	Poi
prap	⌞	prcue	⌞	prec	⌞	precap
PrecedesSlantEqual	⧹	precnapprox	⧹	PrecedesTilde	⧹	precn
prime	′	primes	ℙ	prnap	⧹	prn
Product	∏	profalar	∅	proflne	⌒	profs
Proportional	∝	propto	∝	prsim	⧹	pru
Q						
Qfr	ℚ	qfr	q	qint	∫∫∫	Qoi
Qscr	ℚ	quaternions	ℍ	quatint	∫	que

Unicode character set (R-Z)

The Unicode character set is arranged in the alphabetical order of corresponding entity reference names. We may use the characters like “ β ” directly (by copying and pasting the character) or may opt to reference them with entity reference as “[β](#)” within the presentation token elements.

R						
rAarr	$\Rightarrow$	race	$\Rrightarrow$	Racute	$\acute{R}$	racute
raemptyv	$\emptyset$	Rang	$\gg$	rang	$\rangle$	rangd
raquo	$\gg$	rArr	$\Rightarrow$	Rarr	$\rightarrow$	rarr
rarrbfs	$\rightsquigarrow$	rarrc	$\rightsquigarrow$	rarrfs	$\rightarrow$	rarrhk
rarrsim	$\rightsquigarrow$	rarrtl	$\rightarrow$	Rarrtl	$\rightarrow$	rarrw
ratio	$:$	rational	$\mathbb{Q}$	RBarr	$\rightarrow$	rbarr
rbrace	$\}$	rbrack	$\}$	rbrke	$\}$	rbrksld
rcaron	$\check{r}$	Rcedil	$\check{R}$	rcedil	$\check{r}$	rceil
Rcy	$\mathcal{P}$	Rcedil	$\check{R}$	rdldhar	$\equiv$	rdquo
Re	$\Re$	real	$\Re$	realine	$\mathcal{R}$	realpart
reg	$\text{\textcircled{R}}$	ReverseElement	$\ni$	ReverseEquilibrium	$\rightleftharpoons$	ReverseUpEquilibri
Rfr	$\mathfrak{R}$	rfr	$\mathfrak{r}$	rHar	$\Rightarrow$	rhard
rho	ρ	rhov	ϱ	RightAngleBracket	$\rangle$	Rightarrow
RightArrowBar	$\rightarrow$	RightArrowLeftArrow	$\Leftrightarrow$	rightarrowtail	$\rightarrow$	RightCeiling
RightDownVector	$\downarrow$	RightDownVectorBar	$\downdownarrows$	RightFloor	$\rfloor$	rightharpoondown
rightleftharpoons	$\rightleftharpoons$	rightrightarrow	$\Rightarrow$	rightsquigarrow	$\rightsquigarrow$	RightTee
rightthreetimes	$\times$	RightTriangle	$\triangleright$	RightTriangleBar	$\rhd$	RightTriangleEqu
RightUpVector	$\uparrow$	RightUpVectorBar	$\Uparrow$	RightVector	$\rightarrow$	RightVectorBar
rlarr	$\Leftrightarrow$	rlhar	$\Rightarrow$	rmoust	$\}$	rmoustache
roarr	$\rightarrow$	robrk	$\parallel$	ropar	$\rangle$	Ropf
rotimes	$\otimes$	RoundImplies	$\Rightarrow$	rpar	$)$	rpargt
Rightarrow	$\Rightarrow$	Rscr	$\mathcal{R}$	rscr	$\mathcal{r}$	Rsh
rsquo	$'$	rsquor	$'$	rthree	$\times$	rtimes
rtrif	$\blacktriangleright$	rtriltri	$\blacktriangleleft$	RuleDelayed	$\Rightarrow$	ruluhar
S						
sacute	$\acute{s}$	Sacute	$\acute{S}$	sc	$\succ$	Sc
scaron	$\check{s}$	scaron	$\check{s}$	sce	$\succ$	scE
Scirc	$\hat{s}$	scirc	$\hat{s}$	scnap	$\succ$	scnE
scsim	$\sim$	scy	$\mathcal{C}$	Scy	$\mathcal{C}$	sdot
searhk	$\searrow$	searr	$\searrow$	seArr	$\searrow$	searrow
seswar	$\times$	setminus	$\setminus$	setmn	$\setminus$	sext
sharp	$\sharp$	shchcy	$\mathfrak{u}$	SHCHcy	$\mathfrak{u}$	shcy
ShortLeftArrow	$\leftarrow$	shortmid	$ $	shortparallel	$\parallel$	ShortRightArrow
Sigma	Σ	sigma	σ	sigmav	ς	sim
simeq	$\simeq$	simg	$\simeq$	simgE	$\equiv$	siml
simplus	$\sim$	simrarr	$\rightsquigarrow$	slarr	$\leftarrow$	SmallCircle
smeparsl	$\nparallel$	smid	$ $	smile	$\smile$	smt
softcy	$\mathfrak{b}$	SOFTcy	$\mathfrak{b}$	sol	$/$	solb
Sopf	$\mathbb{S}$	spades	$\spadesuit$	spadesuit	$\spadesuit$	spar

sqcup	$\sqcup$	sqcups	$\sqcup$	Sqrt	$\sqrt{}$	sqsub
sqsubseteq	$\sqsubseteq$	sqsup	$\sqsupset$	sqsupe	$\sqsupseteq$	sqsupset
square	$\square$	Square	$\square$	SquareIntersection	$\sqcap$	SquareSubset
SquareSupersetEqual	$\sqsupseteq$	SquareUnion	$\sqcup$	squarf	$\blacksquare$	squf
Sscr	$\mathscr{S}$	ssetmn	$\setminus$	sstarf	$\star$	Star
straightepsilon	ε	straightphi	φ	Sub	$\subseteq$	sub
subE	$\subseteq$	subedot	$\subseteq\!\cdot$	submult	$\subsetneq$	subnE
subrarr	$\Rightarrow$	subset	$\subset$	submult	$\subsetneq$	subseteq
subsetneq	$\subsetneq$	subsetneqq	$\subsetneqq$	subsim	$\subset\sim$	subsub
succapprox	$\succapprox$	succcurlyeq	$\succcurlyeq$	Succeeds	$\succ$	SucceedsEqual
succeq	$\succcurlyeq$	succnapprox	$\succsim$	succneqq	$\succneqq$	succnsim
Sum	$\sum$	sum	$\sum$	sung	$\mathfrak{S}$	Sup
sup2	$\supset$	sup3	$\supset$	supdot	$\supset\!\cdot$	supdsub
supedot	$\supset\!\cdot$	Superset	$\supset$	SupersetEqual	$\supseteq$	suphsol
supmult	$\supsetneq$	supnE	$\supsetneq$	supne	$\supsetneq$	supplus
supseteq	$\supseteq$	supseteqq	$\supseteqq$	supsetneq	$\supsetneq$	supsetneqq
supsup	$\supsetsup$	swarhk	$\swarrow$	swarr	$\swarrow$	swArr
T						
target	$\oplus$	tau	τ	tbrk	$\top$	tcaron
tcedil	$\mathfrak{t}$	Tcedil	$\mathcal{T}$	Tcy	$\mathcal{T}$	tcy
tfr	$\mathfrak{t}$	Tfr	$\mathfrak{T}$	there4	$\therefore$	therefore
Theta	Θ	thetav	ϑ	thickapprox	$\approx$	thicksim
ThinSpace		thkap	$\approx$	thorn	$\mathfrak{p}$	THORN
TildeEqual	$\simeq$	TildeFullEqual	$\cong$	TildeTilde	$\approx$	times
timesd	$\times$	tint	$\iiint$	toea	$\times$	top
Topf	$\mathbb{T}$	topf	$\mathfrak{t}$	topfork	$\mathfrak{P}$	tosa
triangle	$\triangle$	triangledown	∇	triangleleft	$\triangleleft$	trianglelefteq
trianglerighteq	$\trianglerighteq$	tridot	$\triangle\!\cdot$	trie	$\triangleq$	triminus
trish	$\triangle$	tritime	$\triangle$	Tscr	$\mathcal{T}$	tscr
tshcy	$\mathfrak{h}$	TSHcy	$\mathfrak{H}$	Tstrok	$\mathcal{F}$	tstrok
U						
Uacute	$\acute{U}$	uacute	$\acute{u}$	uarr	$\uparrow$	uArr
ubrcy	$\mathfrak{y}$	Ubrcy	$\mathfrak{Y}$	ubreve	$\breve{u}$	Ubreve
Ucy	$\mathcal{Y}$	ucy	$\mathfrak{y}$	udarr	$\mathfrak{u}$	udblac
ufisht	$\mathfrak{r}$	Ufr	$\mathfrak{U}$	ufr	$\mathfrak{u}$	Ugrave
uHar	$\mathfrak{H}$	uharl	$\upharpoonleft$	uharr	$\upharpoonright$	uhblk
ulcrop	$\ulcorner$	ultri	$\nearrow$	umacr	$\bar{u}$	uml
UnderBrace	$\underbrace{}$	UnderBracket	$\underbrace{}$	UnderParenthesis	$\underbrace{}$	Union
uogon	$\mathfrak{u}$	Uopf	$\mathbb{U}$	uopf	$\mathfrak{u}$	Uparrow
UpArrowBar	$\Uparrow$	UpArrowDownArrow	$\Updownarrow$	UpDownArrow	$\Updownarrow$	Updownarrow
upharpoonleft	$\upharpoonleft$	upharpoonright	$\upharpoonright$	uplus	$\oplus$	UpperRightArrow
Upsi	$\mathfrak{Y}$	Upsilon	Υ	upsilon	υ	UpTee
urcorn	$\urcorner$	urcorner	$\urcorner$	urcrop	$\ulcorner$	Uring
uscr	$\mathfrak{u}$	Uscr	$\mathfrak{U}$	utdot	$\mathfrak{u}$	utilde

utrii	▲	uuarr		uuml	u	Uuml
V						
vangrt	Δ	varepsilon	ε	varkappa	κ	varnothing
varpropto	∝	vArr	↕	varr	↕	varrho
varsubsetneqq	⊊	varsupsetneq	⊋	varsupsetneqq	⊋	vartheta
Vbar	⏟	vBar	≠	vBarv	≠	vcy
vdash	⊢	Vdash	⊨	VDash	⊨	Vdashl
veebar	⊔	veeq	≡	vellip	⋮	Verbar
Vert		VerticalBar		VerticalLine		VerticalSeparator
vfr	⵿	Vfr	ℳ	vltri	◁	vnsb
vopf	⵿	vprop	∝	vrtri	▷	vscr
vsubne	⊊	vsupnE	⊋	vsupne	⊋	vscr
W						
Wcirc	Ŵ	wcirc	ŵ	wedbar	Δ	Wedge
Wfr	ℳ	wfr	ℴ	wopf	ℳ	Wopf
weierp	℘	wreath	⋈	Wscr	ℳ	wscr
X						
xcap	∩	xcirc	◯	xcup	∪	xdtri
xharr		xhArr		Xi	Ξ	xi
xmap		xnis	∋	xodot	⊙	Xopf
xotime	⊗	xrarr		xrArr		xscr
xuplus	⊕	xutri	△	xvee	∨	xwedge
Y						
Yacute	Ŷ	yacute	ŷ	yacy	я	YAcy
ycy	ы	Ycy	Ы	yen	¥	Yfr
yicy	й	Yopf	ℳ	yopf	ℳ	Yscr
YUcy	Ю	yuml	ÿ	Yuml	ÿ	
Z						
Zacute	Ź	zacute	ź	Zcaron	Ž	zcaron
Zdot	Ẑ	zdot	ẑ	zeetrf	ℤ	ZeroWidthSpace
zfr	ℤ	ZHcy	Ж	zhcy	ж	zigrarr